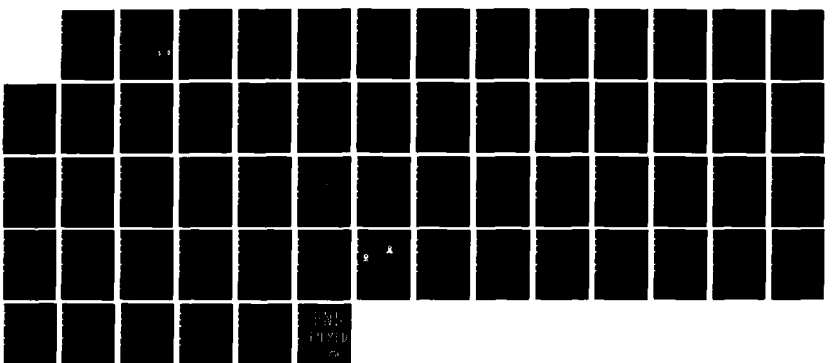
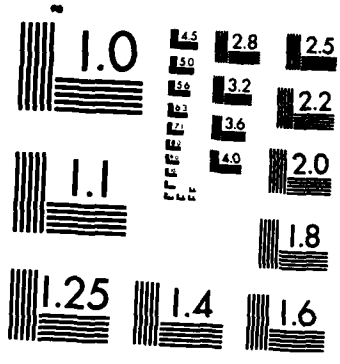
AD-A164 059 HIGH PERFORMANCE PARALLEL COMPUTING(U) TEXAS UNIV AT AUSTIN DEPT OF COMPUTER SCIENCES J C BROWNE ET AL. 1/1

DEC 85 AFOSR-TR-85-1260 F49620-83-C-0049

UNCLASSIFIED F/G 9/2 NL





UNCLASSIFIED

(2)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 85-1260	2. GOVT ACCESSION NO. A164059	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) High Performance Parallel Computing		5. TYPE OF REPORT & PERIOD COVERED Final: 1/1/83 - 12/31/83
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. C. Browne, G. J. Lipovki, M. Malek		8. CONTRACT OR GRANT NUMBER(s) F49620-83-C-0049
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Department and Electrical and Computer Engineering Department The University of Texas at Austin		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304 A3
11. CONTROLLING OFFICE NAME AND ADDRESS Capt. A. L. Bellamy AFOSR/NM Bolling AFB, DC 20332		12. REPORT DATE December 1985
		13. NUMBER OF PAGES 59
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DTIC ELECTE FEB 11 1986 S D D

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The accomplishments of the research project "High Performance Parallel Computing" for the year 1983 span algorithm formulation, parallel programming languages, basic software for the Texas Reconfigurable Array Computer and validation of design concepts for the Texas Reconfigurable Array Computer (TRAC). Image processing, sorting and time dependent partial differential equations were subjects for algorithm formulation and analysis. Accomplishments in parallel programming include: substantial progress toward the implementation of two parallel programming environments, theComputation Structures

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A164 059

FILE COPY

Block 20: ABSTRACT (Continued)

Language, and a task level data flow programming system. The hardware prototype of TRAC made substantial progress towards stability. The state-of-the-art in reconfigurable switch based architectures has been advanced. A result of note is the demonstration of the integration of circuit switching and packet switching in a single interconnection network.

Final Report To The Air Force Office of Scientific Research

On Contract Number F49620-83-c-0049

High Performance Parallel Computing

from
J. C. Browne
Department of Computer Sciences
G. J. Lipovski
M. Malek
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712

10 December 1985

Abstract

The accomplishments of the research project "High Performance Parallel Computing" for the Year 1983 span algorithm formulation, parallel programming languages, basic software for the Texas Reconfigurable Array Computer and validation of design concepts for the Texas Reconfigurable Array Computer (TRAC). Image processing, sorting and time dependent partial differential equations were subjects for algorithm formulation and analysis. Accomplishments in parallel programming include: substantial progress toward the implementation of two parallel programming environments, the Computation Structures Language, and a task level data flow programming system. The hardware prototype of TRAC made substantial progress towards stability. The state-of-the-art in reconfigurable switch based architectures has been advanced. A result of note is the demonstration of the integration of circuit switching and packet switching in a single interconnection network.

1. Research Objectives

This research project was an integrated approach to parallel computation spanning algorithm formulation, programming and software, and hardware/architecture design and prototyping. The goal of the program was to establish a flexible environment for experimental studies of parallel computations. The focus of algorithm formulation and software development research was the Texas Reconfigurable Array Computer (TRAC). It was realized during the course of this year that the concepts being developed for programming and software for TRAC applied to a wider range of architectures. We began to develop our software systems with a broader range of architectures in mind during this year.

The experience of building a prototype of the Texas Reconfigurable Array Computer (TRAC) has proved to be highly rewarding. It has improved our understanding of parallel architectures and, we hope, has made significant contributions to the area of parallel processing.

TRAC employs the novel concepts of reconfigurability and space-sharing in its

organization. These are seen to be the key to the success of the general purpose tightly coupled multiprocessors. Many other unique architectural features were included to enable it to perform equally well in both numeric and non-numeric applications. Most of these features have fulfilled their promise while others have brought to light important issues which may demand further study. The goals for the TRAC reconfigurable architecture may be summarized as follows:

1. Trivially, it must have an organization to accommodate a large number of processors.

2. It must provide for different modes of communication between the processing units.

3. It must have synchronization mechanisms general enough to allow an arbitrary combination of processors to be synchronized.

4. It must be capable of SISD, SIMD, and MIMD modes of execution (12). The system should be dynamically reconfigurable between tasks to support these modes of execution and to maximize the use of system resources.

5. Virtualize the computation. The system must support vertical migration capability and make underlying hardware transparent to the user. A parallel architecture should provide a basis for implementation of parallel languages.

6. Map the architecture to the algorithm. The system organization should be flexible enough so as to be able to mold the architecture to the algorithm, not the algorithm to the architecture as has been applied in the past. It should make available to an algorithm the parallelism that it requests so that its true performance can be evaluated.

7. Give attention to the technology. The machine should be built modularly with a minimum number of unique partitions. This would facilitate the translation of the design into the emerging VLSI technology. This last goal will allow us to assess the engineering decisions that went into the design of TRAC and to document our experience with the TRAC development effort.

2. Research Accomplishments

The research accomplishments of the project during 1983 spanned the range from algorithm formulation to programming languages to operating systems and finally to validation of hardware/architecture design concepts. The subsections which follow give the principle research contributions under each topic.

2.1. Algorithm Formulation

One of the two principal algorithm formulation activities during this year was the establishment of optimal partitioning schemes for time dependent partial differential equations. This work is discussed in a report by E. J. Shipsey [SHI83]. A second algorithm study was study of the relative efficiency of packet switching and circuit switching architectures for realization of histogramming and smoothing algorithms for image processing. A paper on this research was published in the Proceedings of the 1983 International Conference on Parallel Programming [YAS83]. The principal findings are that circuit based architectures will become more efficient as the image resolution and thus the volume of data to be processed increases.

Ramakrishnan and Browne [RAM83], in research performed under the 1982 AFOSR grant of the same title but not reported, developed a paradigm for the design of parallel algorithms for SIMD computer arrays. This research explores the class of algorithms which can be created by combining computational and data movement functions in a single abstract machine instruction. The results obtained included a new algorithm for merging on a bidirectional pipeline of processors.

2.2. Parallel Programming Languages

The year 1983 saw substantial advance in the development of the run-time support system for the Computation Structures Language (CSL). The design for the run-time system of CSL was completed, implemented and partially debugged during this time period. Implementation is taking place on the Dual Cyber 170-750's of the University of Texas Computation Center. A design for a task level data flow language which will complement the capabilities of CSL was initiated during 1983.

The two underlying principles upon which we are basing our parallel programming languages are the separation of representation of dependency relations from computations and the recognition that all parallel computations can be expressed as directional graphs.

The Computation Structures Language is a text string language for specification of computation graphs and for specification of explicit traversal paths for the computation graphs to execute the computation. The task level data flow language uses the same principle of separate units of computation from dependency relations, but implements an implicit traversal of the computation graph.

2.3. Operating System for TRAC

1983 saw the completion of the design for the operating system for the Texas Reconfigurable Array Computer. This research was primarily executed by Mr. Daniel Canas.

Several unique problems arose from this research. The first of these problems was the integration of a virtual memory architecture into the reconfigurable memory structure of TRAC. The capability of the TRAC architecture for switching memory units between processor configurations is a powerful means of sharing of memory.

The mode of moving a memory unit between processor configurations is to generate an interrupt when a processor attempts access to an address which is in a "shared" memory module that is not currently attached to the requesting process. The interrupt service routine realizes its request by establishing a circuit to the memory board holding the requested address. A virtual memory page fault can be created in the same manner.

The result is the establishment of a unification of virtual memory and reconfigurable memory architectures for TRAC. The techniques developed here can be applied to page fault handling in a conventional demand driven page environment if paging is via a switchable memory configuration.

2.4. Hardware Design Concepts

There have been quite a number of accomplishments directly resulting from work during the year on the design and implementation of the prototype. These will be covered subsequently. The memory and the input/output devices are connected to the base nodes of the banyan network. The TRAC architecture distinguishes between the two kinds of devices and has separate interfaces for both. The memory modules employ the Primary Memory Interface while the input/output devices use the Auxiliary Resource Interface (ARI). One significant accomplishment during this year was the definition and implementation of the ARI for support of terminal and disk I/O.

The ARI access has allowed I/O programming to be device-dependent at the user level. The transfers between an Auxiliary Resource (I/O Device) and the primary memory were implemented via descriptor based instructions. The descriptors have the same general format, although their contents are specific to the device being processed. The calling sequences of the instructions are independent of the device being addressed, making the hardware details of the underlying device transparent to the user. Thus the concept of ARI has become central to the virtualization of I/O in TRAC. (The concept is not dissimilar to that of the dev file in UNIX). Also, The actual transfers of data between the device and the primary memory are complementary, allowing transfer of data during every memory cycle. The ARI concept has already been used to connect devices such as terminal (14), printer, disk, self-managed-secondary-memory (15), and the control port (14) to TRAC.

A second major milestone was the successful implementation of the banyan interconnection network. It can be considered to be the most important contribution of the TRAC project to date. It is a two-sided, multistage network with processors at the apex end and memories or input/output devices at the base. It has been built modularly with unique partitioning properties; it has been built using a single building block called the switch module. The switch module itself is easily segmentable and amenable to VLSI implementation.

The primary purpose of a close-coupled computer network is to provide mechanisms for processor-memory and processor-processor communication and those for interprocessor synchronization. The performance of algorithms is directly related to the effectiveness of these mechanisms. Most of the unique characteristics of the TRAC architecture accrue from the capabilities of the supporting interconnection network. The network supports both packet and circuit switched modes of data movement. The packets are essential for implementing an arbitrary permutation on a blocking network while also furnishing asynchronous communication between the processing elements. The packet communication facility provides a means for intra-task data permutation, intertask communication and operating system message interface. The circuit switched modes of interprocessor communication occur in TRAC in the form of *shared* and *instruction* trees. It is believed that the presence of both circuit switched and packet switched modes of communication is necessary to produce the best performance.

It was during this year that prototype validation of both modes of communication was accomplished. This resulted from exhaustive testing of the interconnection network. Programs utilizing both circuit switched shared memory and packet transmission for intra-task communication were successfully executed.

During this period the Control Port was designed and implemented. It is the interface between an arbitrary TRAC processor and the Network Controller which is responsible for the generation of Data, Instruction, and Shared Memory trees. The Control Port was tested and performs at the design goal of 1 MHz (cycle time). All aspects of cold and warm restart were shown to be 100% functional.

The processor microcode space was expanded from 2K to 8K allowing for increased code space along with support for an external arithmetic processing unit (APU) with floating point operations. This restructure provided more flexible microbranch instructions and the removal of wasteful duplicated microcode.

A number of tools were generated in-house to help develop hardware and microcode for TRAC, and were used in addition to the traditional tools such as the high

bandwidth oscilloscopes and logic analyzers. These tools were built around a Z-80 based Cromemco microcomputer. The tools included monitors, testers, and microprogram development aids. These are discussed in more detail in the following paragraphs.

Two monitor programs, MOE and RABUG, were written for the Cromemco System to help develop the hardware and to facilitate debugging of the microcode. A parallel interface was built which allows the Cromemco microcomputer to read and write to the network interface busses and the micro-address busses of all the processor modules. Through separate interface, the micro-computer is able to control the clock. By proper utilization of these interfaces, the monitor programs running on the Cromemco can step through the phases, micro-cycles and memory cycles, and, in addition, can read data from or write data to the busses in the TRAC system.

Monitor program MOE has a capability to sequence through a specified number of phases, micro-cycles, machine cycles, and TRAC instructions, and display or print the data from the busses, and display the contents of the memory pointer registers and the processor status registers for individual processors. To facilitate this, instrumentation was added at the microprogram level to output the required information on the network bus. At the beginning of each instruction, a microprogram routine is executed which supplies the processor status information and memory pointer information to the monitor program, which then displays it on the CRT screen. The monitor program is also capable of receiving its commands from a batch file. The batch files are created to run the machine through an entire program and list bus data or processor status any number of times. This way it is possible to exercise the machine for long periods of time and capture faults if they occur.

Monitor program MOE mentioned above, explicitly controlled the clock, executing considerable Z-80 code for each TRAC clock step. As a result, the TRAC hardware was exercised only at slow speeds. After all functionalities of the architecture were developed and tested, a need was experienced for a more sophisticated monitor which would run the system at the rated speeds of 10kHz, 100kHz or 1 MHz and still retain

the debugging capability. Coupling logic was added to enable the TRAC processors and the Cromemco system to hand-shake and to allow the monitor program RABUG to switch between free running and controlled stepping of the system clock. A capability was also added so that the monitor could send commands, in addition to data, to the memory modules. This latter capability has proved to be helpful for software debugging, since via the monitor, the contents of the memory can be inspected and/or modified. It is now also possible to insert *break points* in the programs to further facilitate their testing. As a result, RABUG now provides a multiprocessor debug facility for both microcode and TRAC machine code program validation.

2.5. Software Development Tools

A software simulator for TRAC was written to help develop system and application software while the hardware was under construction. The software simulator is able to provide the parallel programming environment available on TRAC. It is also able to simulate the shared tree concept and the packet communication. A Pascal compiler, an assembler, and a loader have also been developed for the TRAC system. These programs can also generate code that can be interpreted by the TRAC simulator.

3. Papers Published

[YAS83] Yasrebi, M., Deshpande, S. and Browne, J.C., "A Comparison of Circuit Switching and Packet Switching for Data Transfer in Two Simple Image Processing Algorithms," Proceedings 1983 International Conference on Parallel Processing, Bellaire, Michigan, August 1983.

[RAM83] Ramakrishnan, I.V. and Browne, J.C., "A Paradigm for the Design of Parallel Algorithms with Applications," IEEE Transactions on Software Engineering 9, 1983, pp. 411-415.

[SHI83] Shipsey, E.J., "Computational Organization for Parallel Computation: The Time Evolution of Physical Systems," (in preparation for publication, manuscript attached).

4. Personnel Associated with Project

4.1. Senior Investigators

- J.C. Browne, Principal Investigator
- G. J. Lipovski, Co-Principal Investigator
- M. Malek, Co-Principal Investigator
- R. Jenevein, Consultant
- E. Shipsey, Research Associate

4.2. Graduate Degrees Awarded

- D. Canas, Ph.D., "Operating Systems for Reconfigurable Network Architectured Systems: The Node Kernel," Department of Electrical and Computer Engineering, The University of Texas at Austin, May 1983.
- S. Y. Han, Ph.D., "A Language for the Specification and Representation of Programs in a Data Flow Model of Computation," Department of Computer Sciences, The University of Texas at Austin, May 1983.
- M. Yasrebi, M.S., "A Pipelined Two-Dimensional Fast Fourier Transform Array Processor," Department of Electrical and Computer Engineering, The University of Texas at Austin, May 1983.
- A. Prakash, M.S., "Design and Implementation of an I/O Interface to TRAC," Department of Electrical and Computer Engineering, The University of Texas at Austin, May 1983.

5. Verbal Presentations of AFOSR Sponsored Research

- March 23-26, 1983 - "Modern Parallel Computation Methods," DoD Annual Technical Review on Computer Science and Applied Mathematics, Air Force Academy, Colorado Springs, Colorado.
- April 18-19, 1983 - "Two Paradigms for Parallel Computing," University of Maryland, Department of Computer Science, Distinguished Visitors Program.
- August 1-3, 1983 - Keynote Speech for Conference on Computer Software Performance, Los Alamos National Laboratory, Los Alamos, New Mexico.

- August 17-19, 1983 - "Software for Highly Parallel Architecture," Los Alamos National Laboratory Symposium on Frontiers on Supercomputers.

- April 16, 1983 - "A Language for Highly Parallel Computing," Bell Laboratories, Computer Science Division.

6. Research Project: Perspective

This AFOSR grant was an important element of support for an ambitious comprehensive research program in parallel computation also supported by the Department of Energy and the National Science Foundation. The total result of the project cannot be fully seen from the perspective of only the portion reported herein. There were also six other papers resulting from this project with sponsorship attributed to one of the other granting agencies. The total project, synergizing algorithms, software and hardware was possible only because of the individual contributions of each funding agency to their specific interest areas.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Manuscript: Computational Organization for Parallel Computation: The Time Evolution of Physical Systems		5. TYPE OF REPORT & PERIOD COVERED final: 1/1/83 - 12/31/83 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) E. J. Shipsey (Research Associate)		8. CONTRACT OR GRANT NUMBER(s) AFOSR F49620-83-C-0049
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Department The University of Texas at Austin Austin, Texas 78712		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Capt. A. L. Bellamy AFOSR/NM Bolling AFB, DC 20332		12. REPORT DATE December 1985 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)*		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES paper in preparation		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*		

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The behavior of a physical system as time advances is described by a set of partial differential equations. The state of a system is given when all the functions characterizing its condition are known for all points in space at that given time. Thus the behavior of the system is given by a set of functions defined on some space which is continually changing with time.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

COMPUTATIONAL ORGANIZATION FOR PARALLEL COMPUTATION: THE TIME EVOLUTION OF PHYSICAL SYSTEMS

E. J. Shipsey
Department of Physics
The University of Texas at Austin
Austin, Texas 78712

1. Introduction

The state of a physical system is described, in general, by a set of quantities characterizing its condition. These physical qualities are described mathematically as functions defined on a physical space in which the system is embedded. The behavior of the physical system as time advances is described by a set of partial differential equations. The state of a system is given when all the functions characterizing its condition are known for all points in space at that given time. Thus the behavior of the system is given by a set of functions defined on some space which are continually changing with time.

A more general view may be abstracted from this notion. Instead of time, a more general propagation variable may be considered. If the partial differential equations describing the physical system are defined in terms of N variables, the functions characterizing the system can be viewed in a N-1 dimensional space as the remaining independent variable takes successive values. In this way the system is imagined to evolve in an N-1 dimensional space as viewed by an "observer" travelling along the remaining coordinate axis. Computationally propagation implies that, in principle, if the functions characterizing the physical system are known for some values of the propagation variable they can be computed at a value of the propagation variable which in some sense is further along than the values for which the function is already known. Conceptually and operationally the propagation variable is associated with the unfolding development of the physical system. It may be possible to find more than one propagation variable for a given physical system. The option then arises of selecting a geometry which will give maximum efficiency or stability for a given purpose. An example of such a situation is given in Section IV. Ideally, the propagation variable should be selected so that the computation of the functions characterizing the system is as simple as possible, and that old values of the functions are required at very few other values of the propagation variable.

The computational process is further analyzed by considering the N-1 dimensional subspace (the computational level) which results when the propagation variable is held at a fixed value. The objective is to find, in some sense, regions of computational independence. The independence is of a restricted nature because, of course, the entire space is quite intimately involved in the long time development of the system. An independence of sorts can be conceived for sufficiently short time intervals and of a qualified local nature. This is accomplished by means of partitioning. The computational level is divided into regions which are called computational subdomains. To each computational subdomain is added a border. The border is understood to contain only points contained in an adjacent computational subdomain. The region of the computational level consisting of the computational subdomain plus its associated border is spoken of as a computational domain. The computational domains can be considered to be computationally independent if, for a sufficiently small increment in the propagation variable, all the dependent functions describing the system can be computed for all points in the computational subdomain

at an advanced value of the propagation variable using only their values at points in the computational domain (computational subdomain and its border) at the previous value of the propagation variable. The notion of computational independence thus defined depends on the numerical method adopted as well as the physical description and the mathematical statement of the problem The numerical method selected depends in turn on considerations of numerical stability. The difference in the numerical stability properties of first and second order partial differential equations is the principal reason for the difference in the partitionings adopted for the problems discussed in Sections II and V.

The analysis of the computational process is completed by means of a classification system for the independent variables of the physical system. The three classes of variables are; i) propagation, ii) communication and iii) internal labelling. The first has already been discussed. The last two are dependent upon the mode of partitioning selected. If an independent variable of the system passes from one computational subdomain to another as it takes on all its values it is said to be a communication variable. If, on the other hand, an independent variable of the system which is not a propagation variable remains in a computational subdomain as it takes on all its values, it is an internal labelling variable.

The computer organization for parallel computation follows directly from the partitioning and its associated scheme of variable classification. The computational domains correspond directly to the memories of a set of independent processors. The computation proceeds in steps of one increment of the propagation variable at a time. After each computation is completed data is transferred between memories to fill in the information needed on the borders of the computational domain at the new value of the propagation variable. Each processor is then ready to perform another step of the computation. The process is repeated over and over until the final value of the propagation variable has been attained.

The physical description, mathematical statement and numerical procedure thus produce the computational organization through the mediation of the partitioning process together with its associated scheme of variable classification. The actual work of organization is carried out in the partitioning process. The problem of finding the optimal computational structure for a given formulation of a problem is thus one of finding the optimal partitioning geometry. The partitioning process, however, is only an intermediate step in deriving an efficient computational structure. The search for an optimal computation structure actually begins with the problem formulation itself. Problem formulation can be considered to be comprised of the three aspects; physical, mathematical and numerical. Seemingly minor reformulations can have quite drastic effects on computational structures. A little insight in the initial phases of development can often reap greater efficiencies than the most ingenious geometrical creations in the latter stages. In Section III an example is given in which a slight modification in the mathematical formulation allows parallel computations to be carried out in special cases.

The essential feature of propagation and communication variables is that the numerical problem which must finally be solved must contain very sparse coupling in these variables and be highly explicit. The coupling might be seen in the mathematical formulation but becomes more apparent in the numerical analysis. Some examples of couplings which prevent variables from being communication or propagation variables are given in Section III. In order that partitioning of a computational level be possible it is apparent that only a small region in the communication variable subspace must be involved in the calculation. The most immediate requirement of the numerical system to be solved is that only nearby values (in the directions of the communication variables) be coupled together. In addition, the solution process itself must not spread the calculation into other domains of the partitioning. This latter condition is quite restrictive and actually means that the numerical scheme expresses the functions describing the physical system at one particular point in the computational level in terms of "neighboring" points in earlier computational levels or at only a very specialized class of points in the advanced level. This requirement which arises from considering partitioning in the communication variable subspace actually

furnishes the crucial test of the propagation variable selection. Explicit numerical schemes are not always stable or may require extremely small step sizes to achieve stability, so that the number of systems which can be treated in this way are limited.

The problems most readily organized in this manner belong to the class of problems for the time evolution of physical systems. The equation which will be studied in the next section is the classical Liouville equation. This equation, besides its own intrinsic interest, allows the partitioning procedure to be analyzed in the simplest possible context. Section III deals with the electrostatic Vlasov equation. The Vlasov equation cannot be treated in general by the techniques discussed in the present work. Two special instances are given which allow the mathematical restatement of the problem for present purposes. Laser pulse propagation is discussed in Section IV and a further elaboration of the partitioning procedure introduced. Finally, in Section V the two dimensional diffusion equation which exhibits some difficulties arising from second partial derivatives is described.

2. Nearest Neighbor Communication, The Simple Example of the Classical Liouville Equation

The Liouville equation is a first order homogeneous partial differential equation. Physically the equation describes the evolution in time of a distribution in phase space. That is, if a mechanical system can be described in terms of 3N spatial coordinates, a possible state of the system is represented (classically) by a point in a 6N dimensional space consisting of the 3N spatial coordinates and the 3N associated momenta. The collection of all possible states satisfying some prescribed conditions can be described in terms of a density in the 6N dimensional space, which is called the phase space of the system. The Liouville equation describes the *time* history of a distribution in phase space which has some prescribed form at the initial time. The equation thus represents a first order initial value problem which, aside from its dimensionality, might represent the simplest possible mathematical system.

The classical Liouville equation can be written[1]

$$\partial\rho/\partial t = \Sigma\,(\rho_i/m_i\;\partial\rho/\partial q_i - \partial V/\partial q_i\;\partial\rho/\partial p_i) \tag{1}$$

where i refers to a particular degree of freedom associated with coordinate q_i, momentum p_i and mass m_i. The forces are assumed to be derivable from a potential energy function denoted by V.

A simple example is a harmonic oscillator of one degree of freedom. This system is exactly solvable and furnishes a convenient test case for numerical techniques. The parameterless form of the equation is

$$\partial f/\partial t + y\,\partial f/\partial x - x\;\partial f/\partial y = 0 \tag{2}$$

with solution

$$f(x,y,t) = g(x\cos t - y\sin t,\; x\sin t + y\cos t) \tag{3}$$

where

$$f(x,y,o) = g(x,y) \tag{4}$$

is the initial condition. The initial condition, and the differential equation, is such that the motion is bounded, that is f(x,y,t) also satisfies the boundary condition

$$f(x,y,t) \to 0 \text{ if } |x| \text{ or } |y| \to \infty. \tag{5}$$

A practical numerical technique for first order initial value problems is approximation by a truncated

power series in time,

$$f(t+\delta t) \sim f(t) + \delta t\, \dot{f}(t) + (\delta t^2/2)\, \ddot{f}(t) \tag{6}$$

where dots denote partial differentiation with respect to time. The first derivative with respect to time is obtained directly from the partial differential equation itself, Eq(2), and the second derivative is obtained by differentiating this equation once with respect to time. This procedure can be repeated arbitrarily many times, but the resulting expression becomes too complex to be useful. The n^{th} time derivative will involve n^{th} order spatial derivatives which increases the numerical complexity also. The other partial derivatives are approximated by finite difference procedures. If h is the spacing between grid points, these approximations are, to fourth order in h,

$$\partial f/\partial x = (2/3h)\{f(x+h)-f(x-h) - 1/8[f(x+2h)-f(x-2h)]\} \tag{7}$$

and

$$\partial^2 f/\partial x^2 = 4/3h^2\{f(x+h)+f(x-h) - 1/16[f(x+2h)+f(x-2h)] - 15/8\, f(x)\} \tag{8}$$

as can easily be verified by power series expansion. The cross derivative is obtained by applying Eq(7) twice.

Stability analysis of simple numerical schemes for simple initial value problems [2,3] shows that error propagation will be stable in such schemes, provided

$$v\, \delta t/\delta s < 1 \tag{9}$$

where v is some characteristic velocity and s represents each of the independent variables except time. The procedure above suggests itself as a means of utilizing large enough displacements in the non-temporal variables to obtain numerical stability, while also obtaining numerical accuracy. Its practical success has been verified by numerical computation. An alternative numerical scheme accurate to second order in the variables Δt and Δx is illustrated for the very simple equation

$$\partial u/\partial t = \partial u/\partial x \tag{10}$$

by

$$u(x,t+\Delta t)-u(x,t) = (\Delta t/4\Delta x)\{u(x+\Delta x,t+\Delta t) -u(x-\Delta x,t+\Delta t)+u(x+\Delta x,t)-u(x-\Delta t)\}. \tag{11}$$

The value of u at the points $\{....,x-2\Delta x,x-\Delta x,x,x+\Delta x,x+2\Delta x...\}$ at the new value of time, $t + \Delta t$, now requires the solution of a linear system of equations. This is an example of an implicit scheme, whereas the scheme developed from Eq(6) represents an explicit scheme. The present results seem to be that the explicit scheme described above requires much less computational labor than such implicit schemes.

The explicit scheme adopted moves the function f forward in time with step in time of size δt. The truncated power series (Eq(6)) is used with time derivatives furnished by the partial differential equation itself. The other derivatives required are given finite difference formulas Eqs (7) and (8). Inspection of these finite difference expressions reveals that fewer points can be computed at time $t + \delta t$ than were available at time t. In other words if, for instance, the area of the computational grid is a square, the points on the border and next to the border cannot be computed at the next instance in time. If there are $2n + 1$ non-temporal points initially in each direction this means that after n steps in time the procedure can only supply the function at a single point.

This difficulty is overcome for distributions satisfying Eq. (5) by making the approximate solution zero at the boundary points and at the points neighboring the boundary. At first glance this procedure may seem somewhat arbitrary as, in effect, the normal derivative of the function as well as the function itself is

set equal to zero at the boundary. Numerical studies, however, show that for a sufficiently large domain of approximation no problems arise. The higher order approximation (with the function set equal to zero on the boundary and its neighboring points) has been compared to an approximation using lower order finite difference expressions for the non-temporal derivatives (which requires only that the approximate solution be zero on the boundary itself) and found to be much superior. Distribution functions in both coordinate and momentum variables are always required to vanish as the momentum becomes infinite. (Physical problems with entities moving at infinite velocity usually have no significance.) The techniques described above should always be applicable to the momentum (or velocity) variables. Other boundary conditions may be applied to the spatial coordinates, however. If the distribution function is only strictly defined in a finite spatial domain (reflecting walls for example), clearly the present methods are nonapplicable. Periodic boundary conditions, for example

$$\rho(...q_i+L_i,p_i...) = \rho(...q_i,p_i...) \tag{12}$$

where L_i is the period of the q_i^{th} coordinate can be handled by an obvious modification.

The process of generating an approximate solution to Eq(2) can be envisaged as follows. A region of the x,y plane is selected suitably large that f(x,y,t) can be assumed to be zero on the boundary. A grid with spacing Δx in the x direction and Δy in the y direction is laid out on the region. The approximate function is assigned the value zero on the boundary points and the points neighboring the boundary. The approximation retains these values for all times. The initial values of f(x,y,t) are computed at all the other interior grid points. Using these points which represent f(x,y,t) at time equal zero, f(x,y,t) is computed at time equal δt by means of Eqs(6), (2), (7) and (8). It is clear that many of these last computations are quite independent of each other.

This last step can be partitioned into several independent computations, if the space of computation is divided into subregions with, however, redundant points added to the boundaries as are required by the calculation. This is shown in Fig. 1. The columns of the subregions in Fig. 1 correspond to values of f(x,y,t) for x and t fixed. Time is, of course, constant through the region and x varies in the horizontal direction. The double border of the entire region is shown filled in with zeros, but this need not be done explicitly. The leftmost region (it may be thought of as an independent processor together with an associated memory) provides the means of advancing columns 3 through 6, the center region columns 7 through 12, and the rightmost region 13 through 16. Only one time step can be made since not all the columns in each region are updated in this time step. Accordingly, updated values of f(x,y,t) are moved from subregion to subregion as required to form the boundary and neighboring boundary values in each subregion for the next step. The columns required are shown crosshatched in Fig. 1 and the columns updated which supply these values are shown shaded diagonally in Fig. 1. The data movement is shown by the arrows. The partitioned calculation is seen to proceed in two steps. The first is the time propagation of f(x,y,t) in each independent subregion (which can be carried out simultaneously) and second the transfer of data between subregions to prepare for the next step. The essential feature of the Liouville equation for the one dimensional harmonic oscillator is that data transfer is only needed between adjacent subregions.

A slightly more complicated situation arises from a periodic one dimensional Liouville equation such as

$$\partial f/\partial t + y\ \partial/\partial x + \sin(x)\ \partial f/\partial y = 0. \tag{13}$$

Here the function is periodic in x. The partitioning is shown in Fig. 2. The new feature is the transfer of data from one end of the region to the other. A simplification immediately suggests itself. If the variable y is arranged to vary across the columns, and x varies across the rows, the periodicity will be contained inside each of the subregions of the partition and (with the appropriate modification of the internal conditions) the situation described by Fig. 1 is achieved.

The independent variables of a numerical procedure for solution of a partial differential equation are seen in this partitioning context to have three roles; i) propagation, ii) communication, and iii) internal labelling. In both these examples the time, t, is solely concerned with propagation. In the example of the one dimensional harmonic oscillator as discussed above, x is both a communication variable and an internal labelling variable (since f(x,y,t) is being computed at more than one value of x in each subregion). Finally in the example of the one dimensional harmonic oscillator as discussed above y is solely concerned with internal labelling. As illustrated by the second example of periodic one dimensional motion, reassignment of the variable roles can lead to simpler computational requirements. One of the principal tasks of parallel programming may be said to be that of assigning the optimum role to each variable.

Further subdivision of the computational domain can be considered. In Fig. 3 partitioning in both the x and y directions is shown for the one dimensional harmonic oscillator. The boundary condition is again shown on the extreme borders. The movement of data is again indicated by arrows. No diagonal movement is required if the data is moved in proper sequence. The vertical moves are made first. That is, the rows containing unshaded plus cross hatched areas are moved vertically to the neighboring subregion and form the unshaded and stippled areas of the border. The horizontal moves are made last. The internal diagonal shaded, cross hatched, and recently arrived stippled areas are moved horizontally to form the diagonally shaded areas of the border of the neighboring subregion. The data in the cross hatched area is seen to be moved twice and to fill the corner of the border in the diagonally opposite subregion. By means of this double movement the diagonal movement required to fill the corners of the border has been achieved.

The two dimensional subdivision can be thought of as carried to its ultimate limit if only four points in a square array remain in the computational region. (In Eqs(7) and (8) a five point finite difference scheme is used, if a three point scheme were employed the ultimate limit of the computational area would be a single point.) This is shown in Fig. 4. The data transfer areas have completely coalesced and filled the computational area which is the central square containing four points. The boundary condition is shown by the zeros and the data moves are again indicated by arrows. All the vertical moves are made first and only a single square of four points is moved. All the horizontal moves are made last and each time the central column of non-zero data is moved. In the first sequence of moves the middle squares on the horizontal borders of the subregions are filled and in the second sequence the remaining vertical sides of the subregion borders are filled.

The notion of an ultimate partitioning is useful in classifying a physical problem and indicates just exactly how much parallelism is inherent in a mathematical structure. The idea of an ultimate partition has more utility in more complicated situations. Consider again the general Liouville equation given by Eq.(1). In applications involving large numbers of molecules the dimensionality of the partial differential equation is so large that numerical solution is never attempted. Smaller systems, however, can be of interest. A linear triatomic molecule (with heavy enough atoms to make classical mechanics approximately valid) can be modeled with four variables if bending and coriolis interactions are neglected. The variables consist of two relative spatial variables which determine the separation of the atoms and the two associated momenta. If the two new variables are thought of as internal labelling variables, the computation might proceed as in Fig. 1. The four dimensional problem now requires each processor to perform a very large calculation. To reduce each processor's labor a finer partitioning is required and the situation in Figs. 3 or 4 is considered. The ultimate two dimensional partitioning in Fig. 4 may perhaps be over elaborate for the problem originally considered, but in a higher dimensional problem it may be quite viable. Higher dimensional nearest neighbor networks can be conceived but seem unduly complicated, particularly for dimension greater than three. Finally, if a doubly periodic system is considered (two periodic space variables and their associated momenta) the most natural choices for the communication variables are the two momentum variables, if a two dimensional partitioning as in Figs. 3

or 4 is employed.

3. Nearest Neighbor Communication, the Vlasov Equation in Special Situations

The Vlasov equation is an approximate description of the behavior of a system of free electrons which avoids the high dimensionality of the Liouville equation. The equation for the distribution function f(r,v,t) (in the usual terminology velocity is discussed in place of momentum) is [4,5]

$$\partial f/\partial t + v\cdot\nabla f - e/m\ E(r,t)\cdot\nabla_v f = 0 \tag{14}$$

with

$$E(r,t) = -\nabla \phi\,(r,t) \tag{15}$$

and

$$\nabla^2 \phi = 4\pi e\{ \int dv\ f(r,v,t) - n_i(r)\} \tag{16}$$

where e and m are the charge and mass of the electron, respectively and $n_i(r)$ is the charge density of the positive ions which are assumed to be so slow moving with respect to the electrons that their motion may be neglected. The analogy to the Liouville equation can be made by identifying V in Eq. (1) as $-e\phi$ and taking N equal to one. The three dimensional spatial gradient operator is denoted by ∇ and ∇_v is the gradient in the three dimensional velocity subspace. Another difference between the two equations is that ϕ represents the non local self consistent field of the electrons whereas V in Eq. (1) is strictly local. This feature presents both the numerical difficulties of the Vlasov equation and a nonlinear aspect which gives the equation great physical interest.

The numerical procedure outlined in the last section requires an expression for the second time derivative of f and thus if Eq. (14) is differentiated the time derivative ultimately of $\phi(r,t)$. This is obtained by differentiating Eq. (16) and using Eq. (14)

$$\nabla^2\dot{\phi} = -4\pi e\ \nabla\cdot\int v\ f(r,v,t)dv \tag{17}$$

where the term involving ∇_v has been integrated assuming f(r,v,t) vanishes for large velocity. The time propagation procedure can thus be applied provided ϕ and $\dot{\phi}$ can be obtained from Eqs. (16) and (17). These Poisson equations cannot in general be solved with the partitioning procedures discussed in the last section, and form a specialized area of research in themselves. Similar mathematical systems as represented by Eqs. (14), (15) and (16) arise in fluid dynamics.[6]

Situations do exist, however, which can be adapted to the methods of the last section. If f(r,v,t) and E(r,t) are independent of spatial coordinates y and z the equations can be simplified to

$$\partial f/\partial t + v\ \partial f/\partial x + E\ \partial f/\partial v = 0 \tag{18}$$

and

$$\partial E/\partial x = \int_{-\infty}^{+\infty} dv\ f(x,v,t) - q_i(x) \tag{19}$$

where the components of velocity v_y and v_z have been integrated out of the problem and the independent variables as well as the densities and field have been reparameterized [4]. Differentiating Eq.(20) with respect to time, substituting Eq. (18) and using the fact the f vanishes for large velocity

$$\partial E/\partial t = -\int_{-\infty}^{\infty} v\ f(x,v,t)\ dv + c(t) \tag{20}$$

where c(t) is an integration constant. If the ions are essentially restricted to a finite region which contains zero net charge, E vanishes at infinity (external fields are assumed to be absent) and c(t) vanishes. In periodic situations the assumptions are less clear, but it seems that c(t) can be chosen to be zero, also.

The working system of equations is taken to be Eqs. (18) and (20) (with c(t)=0). It is obvious from Eq. (20) that since v is involved in an integral over its entire range, v cannot be a communication variable in a nearest neighbor communicating partition if computations are to proceed simultaneously. The only partitioning available for the one dimensional Vlasov equation is thus of the kind illustrated by Fig. 1. For systems periodic in x there is no choice but to use the kind of partitioning described by Fig. 2. The only new complication is that for every column of f(x,v,t), x and t fixed and v ranging, a single value of E(x,t) must be added.

In most applications so far the system actually solved consists of Eq. (18) and (19). The reason for this choice is that Eq. (19) involves a lower moment of the distribution than does Eq. (20) and is thus thought to possess more desirable numerical properties. Serial computations using the methods of the previous section, however, have been performed and show no reason for preferring one to the other.

The ultimate partitioning for the one dimensional non periodic Vlasov equation is shown in Fig. 5. The data transfer areas have again coalesced completely into the data computational area. Each different area of the subdomain contains two columns of data, one for each different value of x. Each column contains f(x,v,t) for all values of v and one value of E(x,t). The ultimate partitioning is only one dimensional since with only nearest neighbor data transfer, v cannot be a communication variable.

The present techniques can be applied to another problem which is potentially interesting. If a laser beam of radial symmetry ionizes a column of gas a cylindrically symmetric distribution of ions and electrons will result. To examine this the Vlasov equation is written in cylindrical coordinates

$$\partial f/\partial t + v_r \partial f/\partial r + (v_\phi/r)\partial f/\partial \phi + v_z \partial f/\partial z + (v_\phi^2/r^2 + E_r)\,\partial f/\partial v_r$$

$$- (v_\phi v_r/r - E_\phi)\partial f/\partial v_\phi + E_z \partial f/\partial v_z = 0 \qquad (21)$$

where again the equation has been reparameterized[4] and $v_r, v_\phi, v_z, E_r, E_\phi, E_z$ are cartesian components of vectors in the instantaneous orientation of the unit vectors in the cylindrical coordinate system. The Poisson Equation for the electric field similarly becomes

$$1/r(\partial/\partial r)rE_r + 1/r(\partial/\partial\phi)E_\phi + (\partial/\partial z)E_z = \int_{-\infty}^{+\infty} dv_r \int_{-\infty}^{+\infty} dv_\phi \int_{-\infty}^{+\infty} dv_z f(r,v,t)$$

$$- q_i(r). \qquad (22)$$

Spatial cylindrical symmetry implies f is independent of ϕ and z and E_ϕ and E_z are zero. The distribution function cannot be independent of v_ϕ and v_z since it must vanish when these variables are infinite. From the form of Eq. (21) v_z can be integrated out of the problem, so that all that remains is $f(r,v_r,v_\phi,t)$ and $E_r(r)$.

The system now becomes

$$\partial f/\partial t + v_r \partial f/\partial r + (v_\phi^2/r^2 + E_r)\partial f/\partial v_r - (v_\phi v_r/r)\,\partial f/\partial v_\phi = 0 \qquad (23)$$

and

$$(\partial/\partial r)\, r\, E_r = r\{ \int_{-\infty}^{+\infty} dv_r \int_{-\infty}^{+\infty} dv_\phi\, f(r,v_r,v_\phi,t) - q_i(r)\}. \qquad (24)$$

The current equation is obtained by integrating this from o to r, requiring E_r to be finite at r=0, differentiating with respect to time, inserting Eq.(23) and integrating by parts, so that

$$\partial E_r/\partial t = \int_{-\infty}^{+\infty} dv_r \int_{-\infty}^{+\infty} dv_\phi v_r f(r,v_r,v_\phi,t). \tag{25}$$

Finally, Eq. (23) can be simplified by replacing the pair of variables r,v_ϕ by the pair r',L where (formally)

$$r' = r$$
$$L = rv_\phi \tag{26}$$

Using the chain rule, and dropping the prime from the resulting equation gives

$$\partial f/\partial t + v_r \partial f/\partial r + (L^2/r^3 + E_r)\partial f/\partial v_r = 0. \tag{27}$$

The "force" which appears in this equation is the "centrifugal force" plus the electrostatic force. The angular momentum is L. Since, classically, the centrifugal force keeps particles away from the axis, for non zero values of L, $f(r,v_r,v_\phi,t)$ is required to be zero at r equal to zero.

The system of equations to be solved consists of Eqs. (25) and (27). Due to the integral in Eq. (25), v_r and v_ϕ (or L) can only be internal labelling values. Again t is the propagation variable and r is the communication variable. A linear partitioning as in Figs (1) or (5) is the only choice. The columns in these figures are replaced by matrices with v_r and v_ϕ (or L) varying along the rows and columns and with each matrix is associated a single value of $E_r(r,t)$.

It is interesting at this point to compare the four variable Liouville equation discussed in the last section on the partitioning given in Fig. 4 with the cylindrical Vlasov equation just discussed on the partitioning given in Fig. 5. These configurations represent using the ultimate partitioning for the next most simple problem in the new context. For the Liouville equation the computational area of the simplest problem consists of four points. In the new context each of these points is replaced by a matrix whose rows and columns can be thought of as generated by the two new variables. For the Vlasov equation the computational area of the simpler problem consists of two columns which are generated by variation in a velocity variable (along with a single value of the electric field). In the new context these columns are replaced again by matrices whose elements are generated by varying the values of the two velocities. In these two cases the matrices are roughly comparable in size, the difference being that there are four in the case of the Liouville equation considered and two in the case of the cylindrical Vlasov equation. The computational labor required in each subdomain is thus of the same order of magnitude. Again the reason for these numbers, four and two, is because the high order finite difference schemes given by Eq. (7) and (8) are being considered. If a three point scheme was employed only one matrix would be required.

4. Numerical Organization of Laser Propagation Computations

The physical situation is that of a laser pulse of radial symmetry perpendicular to its direction of propagation passing through a medium containing matter which the laser pulse is capable of exciting. A simple mathematical model is given by the three equations [7,8,9]

$$\{\partial/\partial z + (1/c)\partial/\partial t - ia(\partial^2/\partial r^2 + (1/r\ \partial/\partial r)\}E = bP, \tag{28}$$

$$(\partial/\partial t + k_p + i\alpha)P = \beta EW, \tag{29}$$

and

$$(\partial/\partial t + k_w)(W - W^\circ) = \gamma(EP^* + E^*P). \tag{30}$$

The independent variables are the time t and the cylindrical coordinates z and r. The Pulse is propagated along z and r is the distance (perpendicular to z) from the pulse center. The electric field due to the laser pulse, E, the polarization of the medium, P, and the inversion density, W, are the dependent variables. Note that E and P are complex. The velocity of light in the medium is c. If the field were suddenly turned off, as seen from Eqs. (29) and (30) P and $W-W^\circ$ would decay with decay constants k_p and k_w

respectively, and W^o is the steady state value of W. The parameters a,b,α,β and γ are combinations of other physical parameters and constants of the system (frequency, π, Planck's Constant etc.) which have been combined together for simplicity. Detailed discussions of the underlying physical phenomena are available. [10,11] Instead of describing the rapidly varying electric field and polarization themselves, E and P actually describe the assumed slowly varying envelopes, and thus some second derivatives have been neglected. The geometrical nature of the initial conditions requires specialized numerical techniques.

New variables τ and η are defined by

$$\tau = t - z/c$$
and
$$\eta = z \tag{31}$$

From the chain rule it is seen that

$$\partial/\partial t = \partial/\partial\tau$$
and
$$\partial/\partial z + (1/c)\partial/\partial t = \partial/\partial\eta. \tag{32}$$

The laser pulse enters the medium at z equals zero at time t equals zero and moves with velocity c. At a given point, z, τ is thus seen to be the time after the "leading edge" of the pulse has passed. A fixed value of τ determines the path in the t,z plane of a disturbance due to the laser pulse. The derivative along the path of such a disturbance is $\partial/\partial\eta$. The transformation does not change the form of Eqs. (29) and (30), so these can be taken over unchanged, provided that it is understood the time is measured from the "leading edge" of the disturbance. The remaining equation, Eq. (28) becomes

$$\partial E/\partial\eta - ia(\partial^2/\partial r^2 + 1/r\ \partial/\partial r)E = bP. \tag{33}$$

The initial condition for E is prescribed at z=0, (η=0)

$$E(o,r,t) = E_o(r,t). \tag{35}$$

The medium is assumed to have its spatially independent steady state values ahead of the laser pulse, so that

$$W(\eta,r,\tau) \to W_o \text{ as } \tau \to o$$
and
$$P(\eta,r,\tau) \to P_o \text{ as } \tau \to o. \tag{36}$$

The situation is depicted schematically in Fig. 6, neglecting for this purpose the variable r. The time history of the laser pulse at z=0 is sketched to the left where the "leading edge" starts at t=0. The "leading edge" propagates along the line labelled τ=0 in the figure. The area below this line has not yet encountered the laser pulse, and thus the initial conditions for W and P are applied on the skewed line τ=0. The boundary condition for the field is applied at z=0, which is the vertical axis to the left labelled t=τ(o). The boundary of the space-time region is indicated by the diagonal shading attached to the lines. The derivative $\partial/\partial\eta$ is taken along the skew lines in the figure with τ constant. Thus P and W propagate in the vertical direction, starting from the line τ equals zero, and E propagates in the skewed direction (τ=constant) starting from the line z equals zero.

The nonorthogonal coordinate system can be simply understood by considering the variation in E along the line τ=constant,

$$\Delta E_\tau = (\partial E/\partial Z)\ \delta Z_\tau + (\partial E/\partial t)\ \delta t_\tau \tag{37}$$

where the subscript τ indicates the special nature of the variation. From Eq. (31)

$$\delta t_\tau = 1/c\ \delta Z_\tau. \tag{38}$$

Substituting this in the preceding equation

$$\Delta E_\tau = (\partial E/\partial Z + 1/c\ \partial E/\partial t)\ \delta Z_\tau. \tag{39}$$

In view of Eq. (32) this can be written

$$\Delta E_\tau = (\partial E/\partial \eta)\, \delta Z_\tau. \tag{40}$$

It is clear from this expression that

$$\delta \eta = \delta Z_\tau \tag{41}$$

as is obtained from Eq. (31) also. It might have been supposed that the arc length along the line τ=constant should have been used, but this is seen not to be the case.

The numerical solution has been given by predictor-corrector[11] and semi-implicit predictor-corrector [9] techniques. These procedures allow a single step in E along τ=constant, and a single step in P and W along z=constant to be taken. These two independent steps must end at the same point if the information required for the next step is to be available. The nature of the differential equations and their boundary conditions have thus provided certain restrictions in the sequence in which the solution is generated. Several different sequences can be conceived, but not all are suitable for partitioning into semi-independent computational procedures. A useful sequence and two associated partitionings are shown in Figs. 7 and 8. The figures are shown on the problem space shown in Fig. 6. The circles indicate the points at which the numerical solution is generated and the number in the circle indicates the step number in which the numerical solution is acquired. Each circle has circles containing smaller numbers (from earlier steps) to the left along τ=constant and below along z=constant. Each circle containing the same number can be computed independently of all the other circles with the same number, and therefore partitioning can be performed across diagonals containing the same number. Two different partitioning are shown, one in Fig. 7 to generate the solution behind the leading edge of the pulse, the other in Fig. 8 to generate the solution neighboring the entrance window to the medium. The boundary region is only shown for the entire problem domain. Only the new values generated are shown in the interior subdomains, their boundary conditions and data transfers are omitted for simplicity. In order to achieve this partitioning the computation is required to have a staggered start. In the first step computation takes place only in subdomain one, in step two parallel computations take place in subdomains one and two, in step 3, computations in subdomains one, two, and three and so on until all subdomains are active.

So far the radial variable r has not been discussed as it clearly does not present any new problems, or enter into the complexities discussed above. The radial variable clearly is not a propagation variable. If the partitioning indicated in Fig. 7 and 8 is implemented in the simple manner of Figs 1 or 5, r is just an internal labelling variable and merely turns the points indicated in Figs 7 and 8 into the columns of Figs 1 or 5. The partitioning can be further refined, however, and r assigned the role of a communication variable to produce a scheme such as is given in Figs 3 or 4.

Finally it is interesting to note that in comparing Figs. 7 and 8 the roles of t and z are interchanged in the two figures. In Fig. 7, each subdomain contains a particular value of τ which remains the same as the solution is developed. Each successive step advances the subdomain to a new value of z (or η). Thus z (or η) may be said to be the propagation variable and t (or τ) the communication variable. Each subdomain in Fig. 8, on the other hand contains the same value of z for all steps and is advanced in steps of τ (or t). Thus in Fig. 8 the propagation variable is t (or τ) and the communication variable is z (or η).

5. Two Dimensional Diffusion

Isotropic diffusion in two dimensions is described by the equation

$$\partial f/\partial t = \partial^2 f/\partial x^2 + \partial^2 f/\partial y^2 \tag{42}$$

when suitable units are chosen for the independent variables. The dependent variable f can be regarded as the concentration of some substance which is initially present in some non-uniform distribution. Only the simplest boundary conditions, that for all time f has some fixed value on a rectangular boundary in the x-y plane, will be considered.

If an explicit numerical approximation such as was employed in Sections 2 and 3 is utilized, the condition for numerical stability will be

$$D\,\Delta t/(\Delta x)^2 < 1 \text{ for } \Delta x \sim \Delta y \tag{43}$$

where D is some characteristic numerical constant. This condition is strikingly different than the condition for first order systems given by Eq. (9). Consider the case in which a numerical result has been obtained which appears physically reasonable. If all the variable spacings are shrunk uniformly the approximation for the first order system will remain stable (if initially stable) whereas a similar kind of approximation for a second order system may become quite unstable. In such an event a perhaps unreasonably small value of Δt may be required for computation. An unconditional stable explicit numerical scheme is required to properly handle second order terms.

The variable transformation

$$\begin{aligned} S &= (y+x)/\sqrt{2} \\ t &= (y-x)/\sqrt{2} \end{aligned} \tag{44}$$

is first carried out. This leaves the partial differential equation unchanged, i.e.

$$\partial f/\partial t = \partial^2 f/\partial s^2 + \partial^2 f/\partial t^2. \tag{45}$$

The two dimensional version [12,13] of a scheme originally devised by Saul'Yev [14] is employed with the s and t variables. An apparently more accurate version of the procedure is available, [15] as well as another application to a more general one dimensional problem [16]. The method is generally called the alternating direction explicit procedure (ADEP). Two different forms of the finite difference expression are given,

$$\begin{aligned} \hat{u}^{\ell+1}_{m,n} - \hat{u}^{\ell}_{m,n} = (\Delta t/2h^2)(&\hat{u}^{\ell+1}_{m-1,n} - 2\,\hat{u}^{\ell+1}_{m,n} + \hat{u}^{\ell+1}_{m,n-1} \\ &+ \hat{u}^{\ell}_{m+1,n} - 2\,\hat{u}^{\ell}_{mn} + \hat{u}^{\ell}_{n,n+1}) \end{aligned} \tag{46}$$

and

$$\begin{aligned} \hat{v}^{\ell+1}_{m,n} - \hat{v}^{\ell}_{m,n} = (\Delta t/2h^2)(&\hat{v}^{\ell}_{m-1,n} - 2\hat{v}^{\ell}_{m,n} + \hat{v}^{\ell}_{m,n-1} \\ &+ \hat{v}^{\ell+1}_{m+1,n} - 2\,\hat{v}^{\ell+1}_{m,n} + \hat{v}^{\ell+1}_{m,n+1}) \end{aligned} \tag{47}$$

where

$$\begin{aligned} t &= l\Delta t \\ s &= n\,h\,\sqrt{2} \\ t &= m\,h\,\sqrt{2}\,. \end{aligned} \tag{48}$$

The finite difference expression for u furnishes an approximation for f at the gridpoint m,n in terms of values of m-1,n and m,n-1 for the same time level plus values from the previous time level. Thus for a given time level f is generated starting from the lower left hand corner. The situation is reversed for v. Taken together, the two equations furnish a means of generating an approximation to f, first in one direction, then in the opposite direction. Two different means of implementing the approximation are in use. The first is an averaging procedure.

$$\begin{aligned} \hat{u}^{\ell}_{m,n} &= f^{\ell}_{m,n} \\ \hat{v}^{\ell}_{m,n} &= f^{\ell}_{m,n} \end{aligned} \tag{49}$$

and

$$f^{\ell+1}_{m,n} = 1/2(\hat{u}^{\ell+1}_{m,n} + \hat{v}^{\ell+1}_{m,n}).$$

The second implementation merely takes alternate directions in subsequent steps,

$$\begin{aligned} \hat{u}^{\ell}_{m,n} &= f^{\ell}_{m,n} \\ \hat{v}^{\ell+1}_{m,n} &= \hat{u}^{\ell+1}_{m,n} \\ f^{\ell+2}_{m,n} &= \hat{v}^{\ell+2}_{m,n} . \end{aligned} \tag{50}$$

The procedure is adapted for partitioning by transforming the expression back to the original variables x and y. Substituting s and t from Eq. (48) into Eq. (44) and solving for x and y gives

$$\begin{aligned} x &= jh \\ y &= kh \end{aligned} \tag{51}$$

where

$$j = n-m$$

and

$$k = n+m . \tag{52}$$

For these variables

$$\begin{aligned} u^{l+1}_{jk} - u^{\ell}_{jk} &= (\Delta t/2h^2)(u^{\ell+1}_{j-1,k-1} - 2\,u^{\ell+1}_{j,k} + u^{\ell+1}_{j+1,k-1} \\ &+ u^{\ell}_{j-1,k+1} - 2\,u^{\ell}_{j,k} + u^{\ell}_{j+1,k+1}) \end{aligned} \tag{53}$$

and

$$\begin{aligned} v^{l+1}_{jk} - v^{\ell}_{jk} &= (\Delta t/2h^2)(v^{\ell}_{j-1,k-1} - 2\,v^{\ell}_{j,k} + v^{\ell}_{j+1,k-1} \\ &+ v^{\ell+1}_{j-1,k+1} - 2\,v^{\ell+1}_{j,k} + v^{\ell+1}_{j+1,k+1}) . \end{aligned} \tag{54}$$

These last expressions allow the calculations for the new time level to be carried out in a vertical rather than diagonal direction. The procedure is shown schematically in Figure 9. In this figure each grid point in the finite difference scheme is in the center of a small square. Only half of the grid points are coupled together by the finite difference expressions (Eqs. (53) and (54)), so that f need only be found at half the grid points. The grid points omitted from the calculation are located in the shaded small squares. Computation of u from Eq. (53) proceeds from the bottom of the figure to the top. One particular step in the computation of u is shown at the bottom of the figure. The grid points corresponding to the $l+1^{st}$ time level in Eq. (53) are indicated by the open circles. The grid points corresponding to the l^{th} level are indicated by solid circles. The calculation proceeds in the direction indicated by the arrow once a horizontal row has been completed. The computation of v as given by Eq. (54) is similarly shown at the top of the figure. This computation proceeds from the top to the bottom of the figure.

The corresponding partitioning and communication for parallel computation is shown in Figure 10. The situation is quite similar to that of Figure 1 except that in the present case data is transferred at the completion of the calculation of each horizontal row rather than at the end of the calculation of the particular time level of the entire domain. The data transferred in the present case is only that of a single grid point instead of the transfer of an entire vertical column in Figure 1. The domain border points containing boundary values for the domain computations are located in the crosshatched area (except for the two rows shown in more detail). Two rows of the finite difference grid are shown in the middle of each domain. The grid points omitted from the calculation (the shaded small squares in Figure 9) are located in the stippled squares. The movement of newly computed data to the boundary of the

neighboring domain (to supply information needed to compute the next row) is shown by arrows.

The time variable (index l in Eqs. (53) and (54)) is a propagation variable in this scheme. The index j corresponds to communication and k to internal labelling. In view of the way the computation proceeds by horizontal rows, k can be also regarded as an internal propagation variable. The independent variables x and y are equivalent in Eq. (42), and essentially equivalent in Eq. (2). In the latter case a two dimensional partitioning is possible. Only a one dimensional partitioning, by contrast, is possible in the present case due to the numerical procedure.

Acknowledgements

Discussion with Mr. Al Rosenberger concerning laser pulse propagation is gratefully appreciated, as well as the advice and encouragement of Professor J. C. Browne. This research was supported by the Air Force Office of Scientific Research under Grant Number AFOSR F49620-83-C-0049.

References

1. Landau, L.D. and Lifshitz, E.M., Statistical Physics (Addison-Wesley, Reading, MA, 1958) p. 10.
2. Potter, D., Computational Physics (Wiley, New York, 1973).
3. Richtmyer, R.D. and Morton, K.W., Difference Methods for Initial-Value Problems, (Interscience, New York, 1967).
4. Knorr, G., Z. Fur Naturforsch, *16a*, 1320 (1961).
5. Kellogg, P.J., Physics of Fluids, *8*, 102 (1962).
6. Ref. 2, Chapter 9.
7. Mattar, F.P. and Newstein, M.C., Comp. Phys. Comm. *20*, 139 (1980).
8. Wright, N. and Newstein, M.C., Optics Comm. *9*, 8 (1973).
9. Mattar, F.P., Gibbs, H.M., McCall, S.L., and Feld, M.S., Phys. Rev. Lett., *46*, 1123 (1981).
10. Lamb, G.L., Jr., Rev. of Mod. Phys. *43*, 99 (1971).
11. Icsevgi, A., Lamb, W.E., Jr., Phys. Rev. *185*, 517 (1969).
12. Barakat, H.Z. and Clark, J.A., J. of Heat Transfer *88*, 421 (1966).
13. Larkin, B.K., Math. Comp. *18*, 196 (1964).
14. Saul'Yev, V.K., Integration of Equations of Parabolic Type by the Method of Nets (trans., Tee, G.J., MacMillan, New York, 1964).
15. Liu, S.L., Al Ch E J *15*, 334 (1969).
16. Satter, A., Shum, Y.M., Adams, W.T. and Davis, L.A., Soc. Pet. Eng. J *20*, 129 (1980).

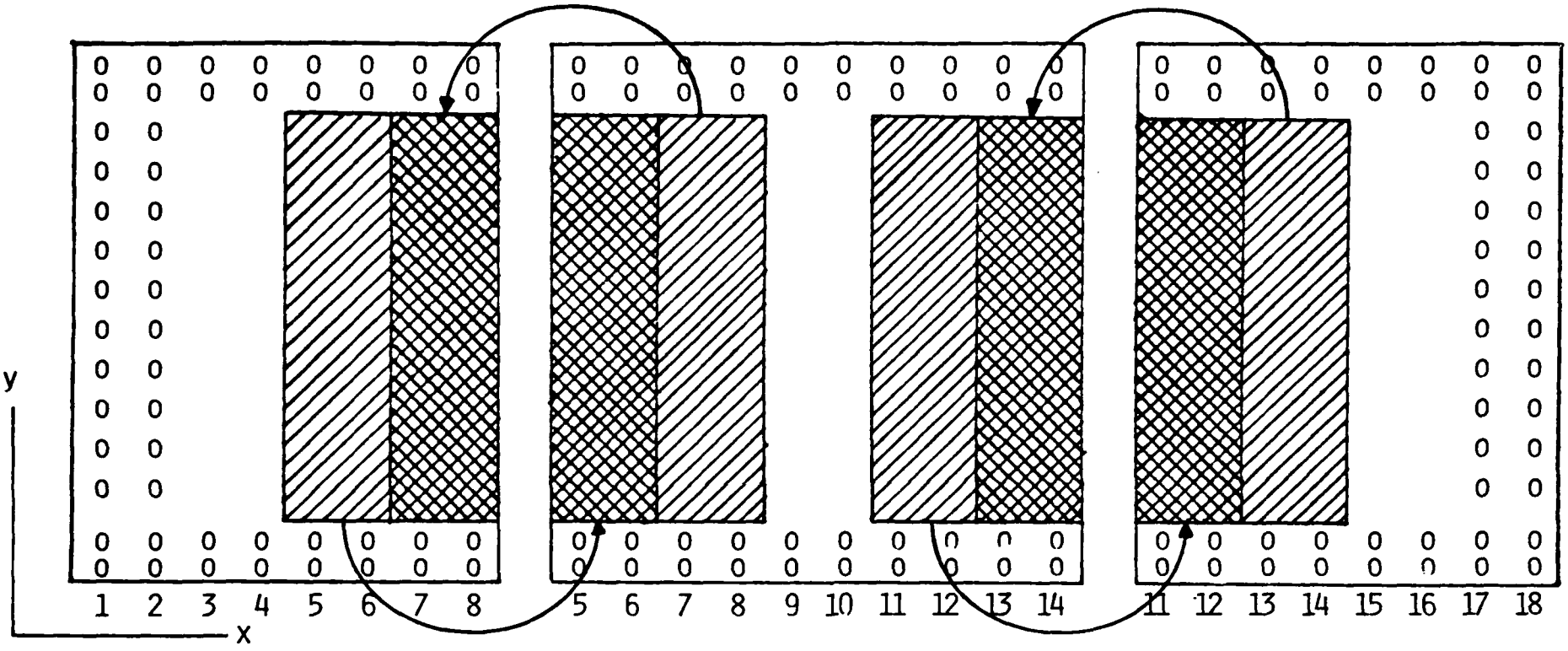


Figure 1. Parallel Processor Organization for Solution of the Liouville Equation for the One Dimensional Harmonic Oscillator

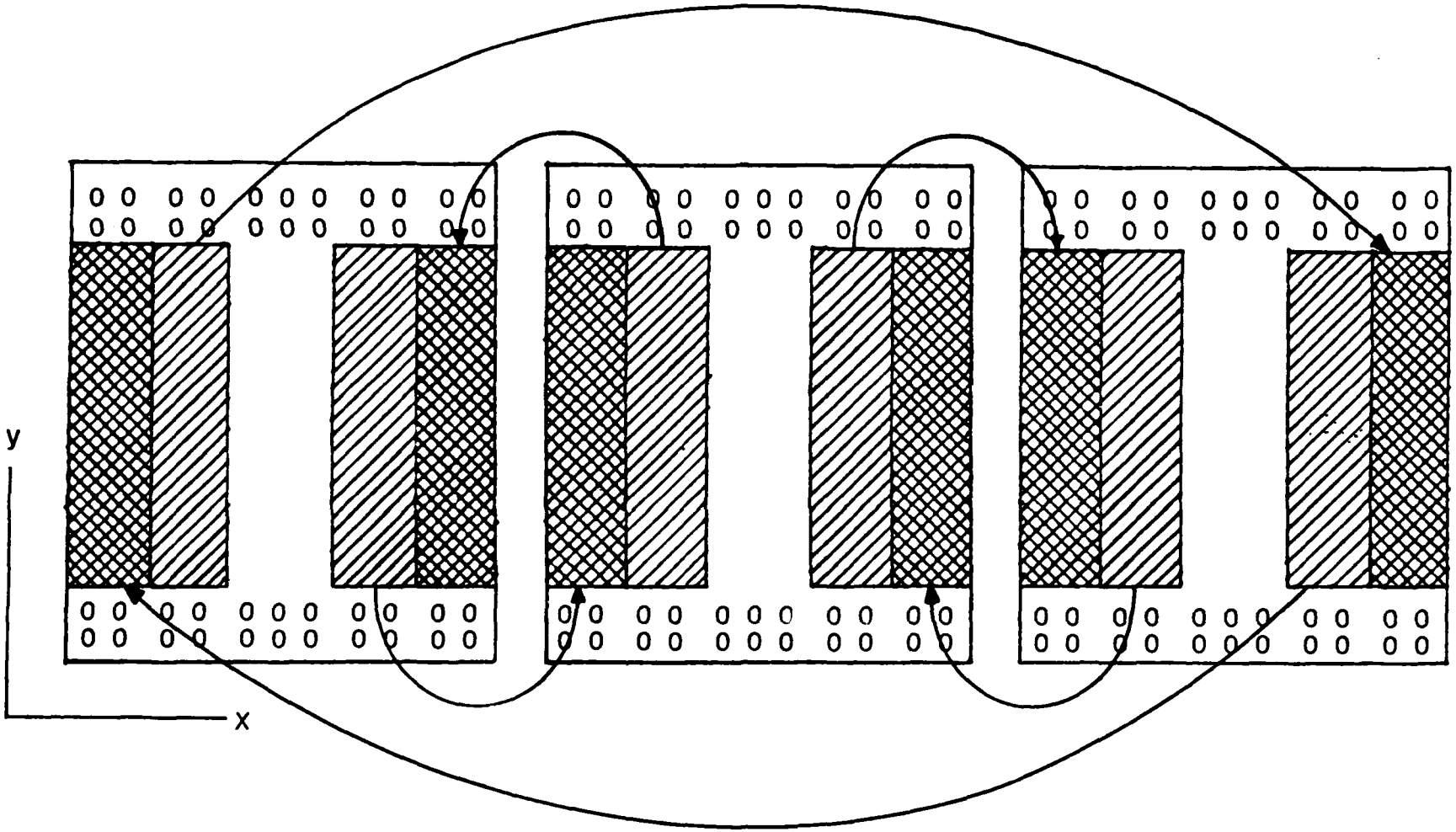


Figure 2. Parallel Processor Organization for Solution of the Periodic One Dimensional Liouville Equation

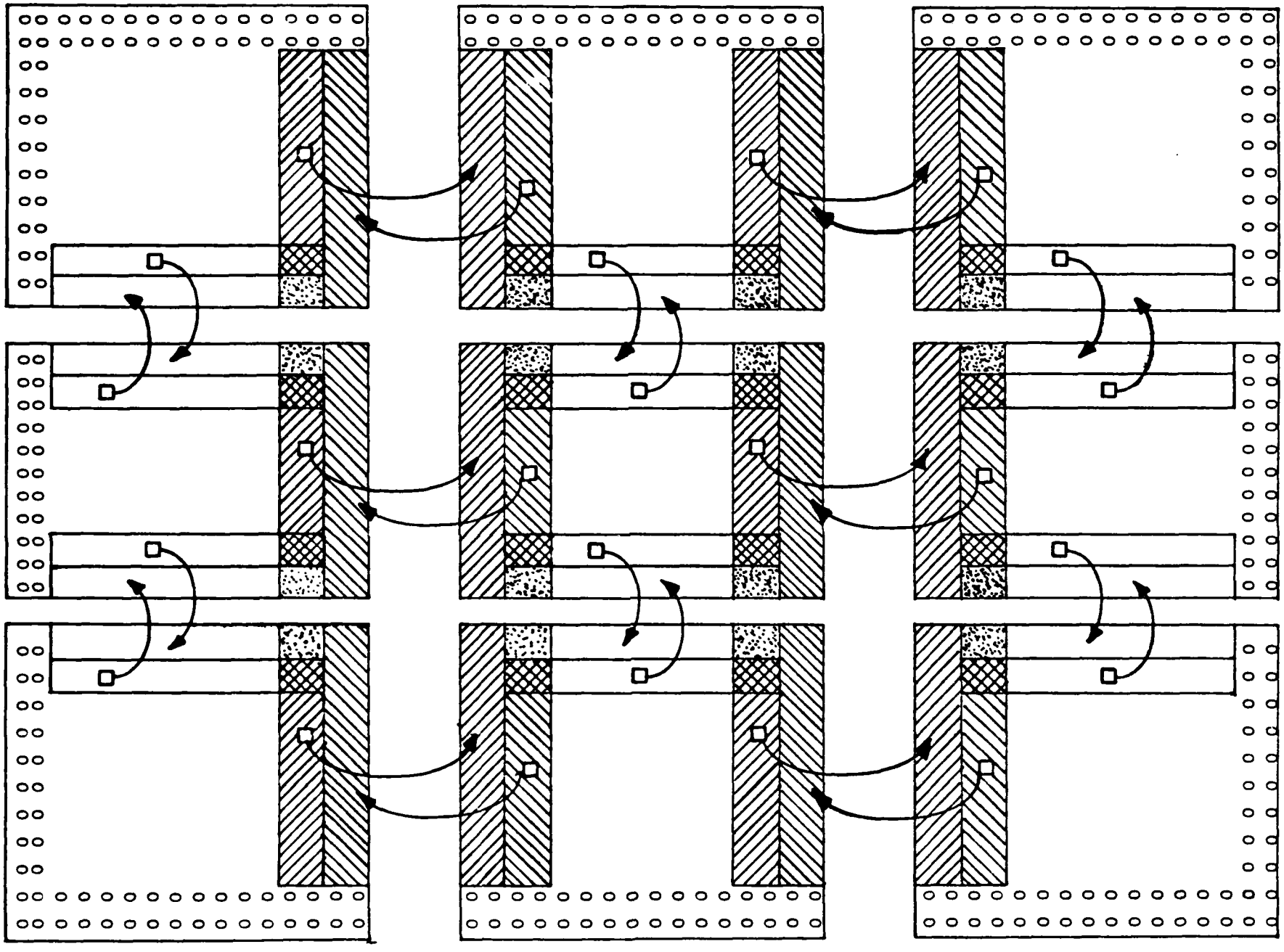


Figure 3. Two Dimensional Organization of Parallel Processors for the Solution of the Liouville Equation for the One Dimensional Harmonic Oscillator

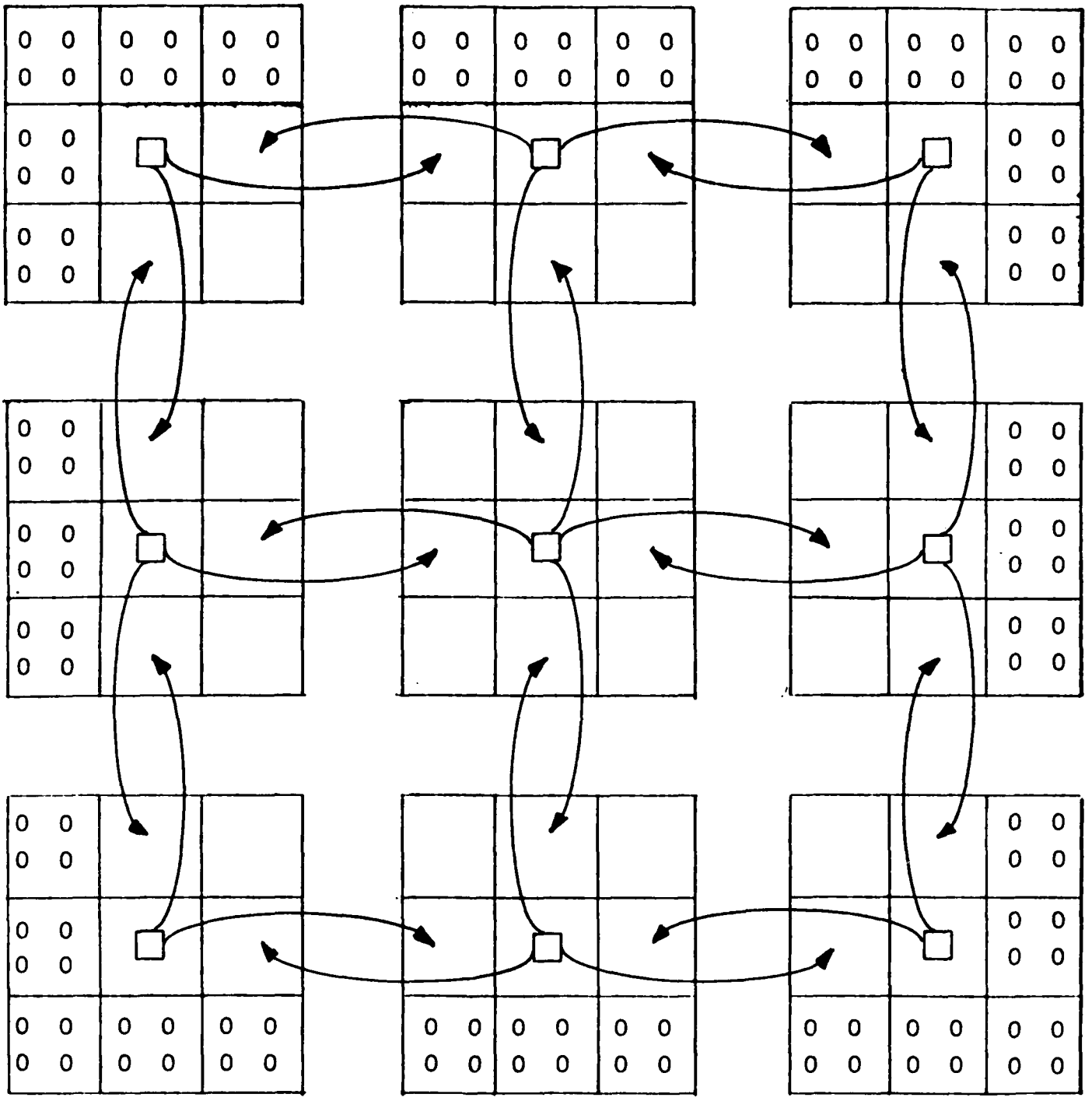


Figure 4. Same as Figure 3 Except That Each Computation Consists Only of 4 Points, the Minimum Number, and That the Data Transfer Blocks Completely Overlap

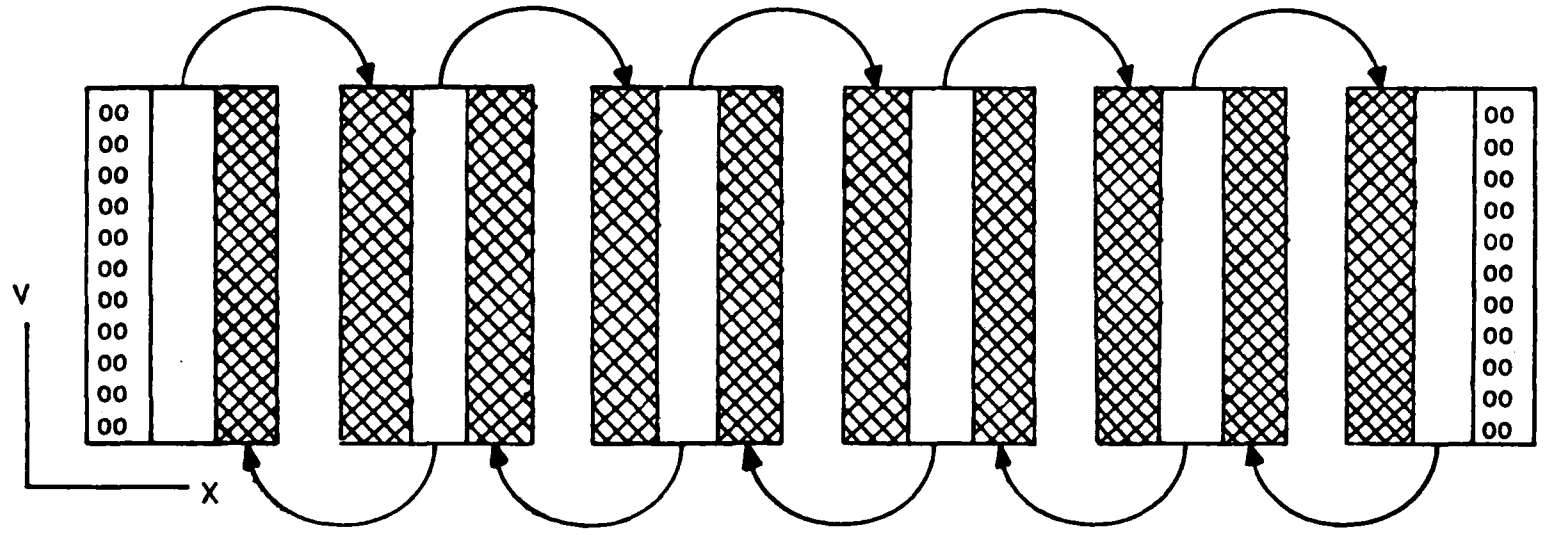


Figure 5. The Finest Partitioning for the
One Dimensional Non Periodic Vlasov Equation

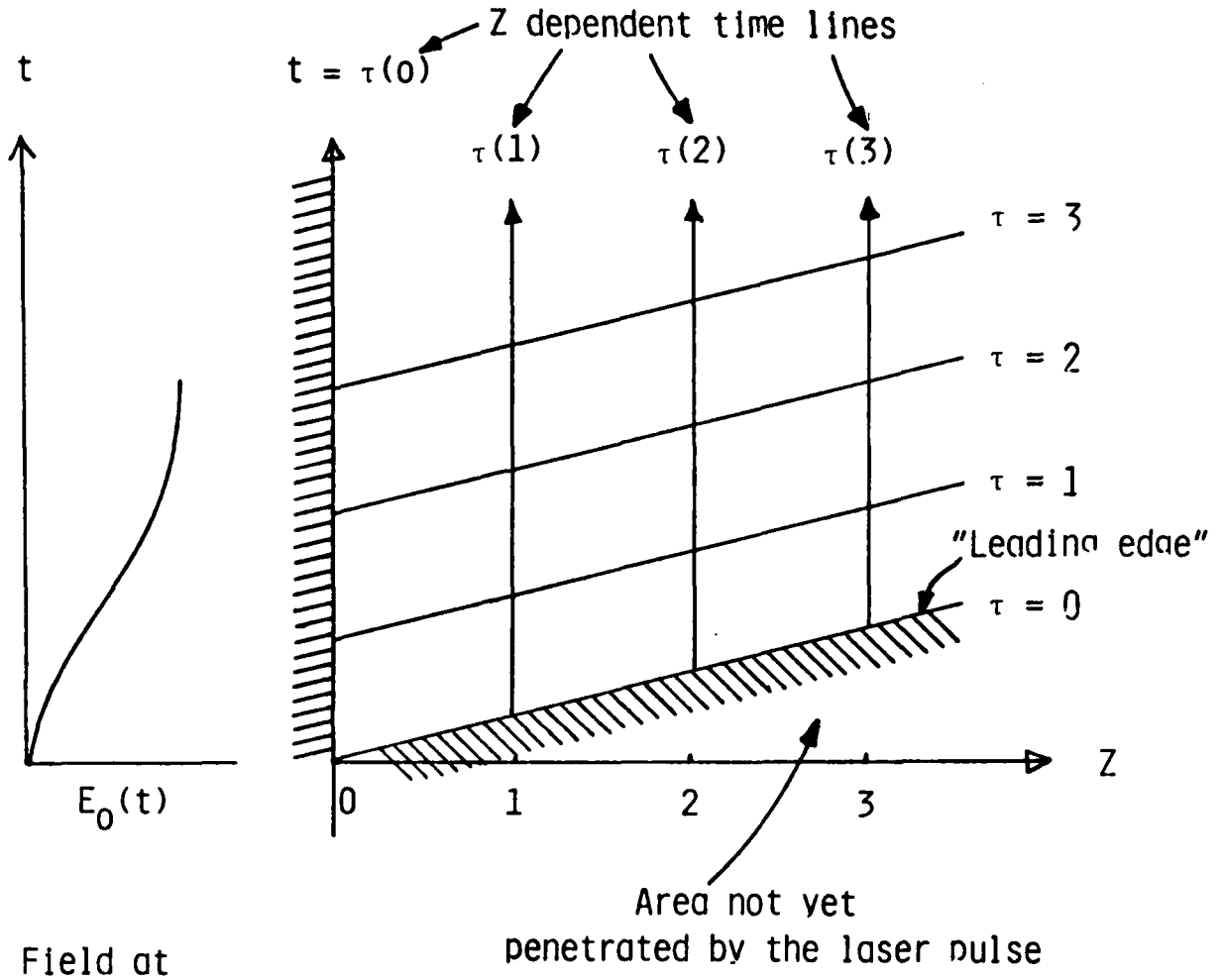


Figure 6. Space-Time Geometry
for Pulse Propagation

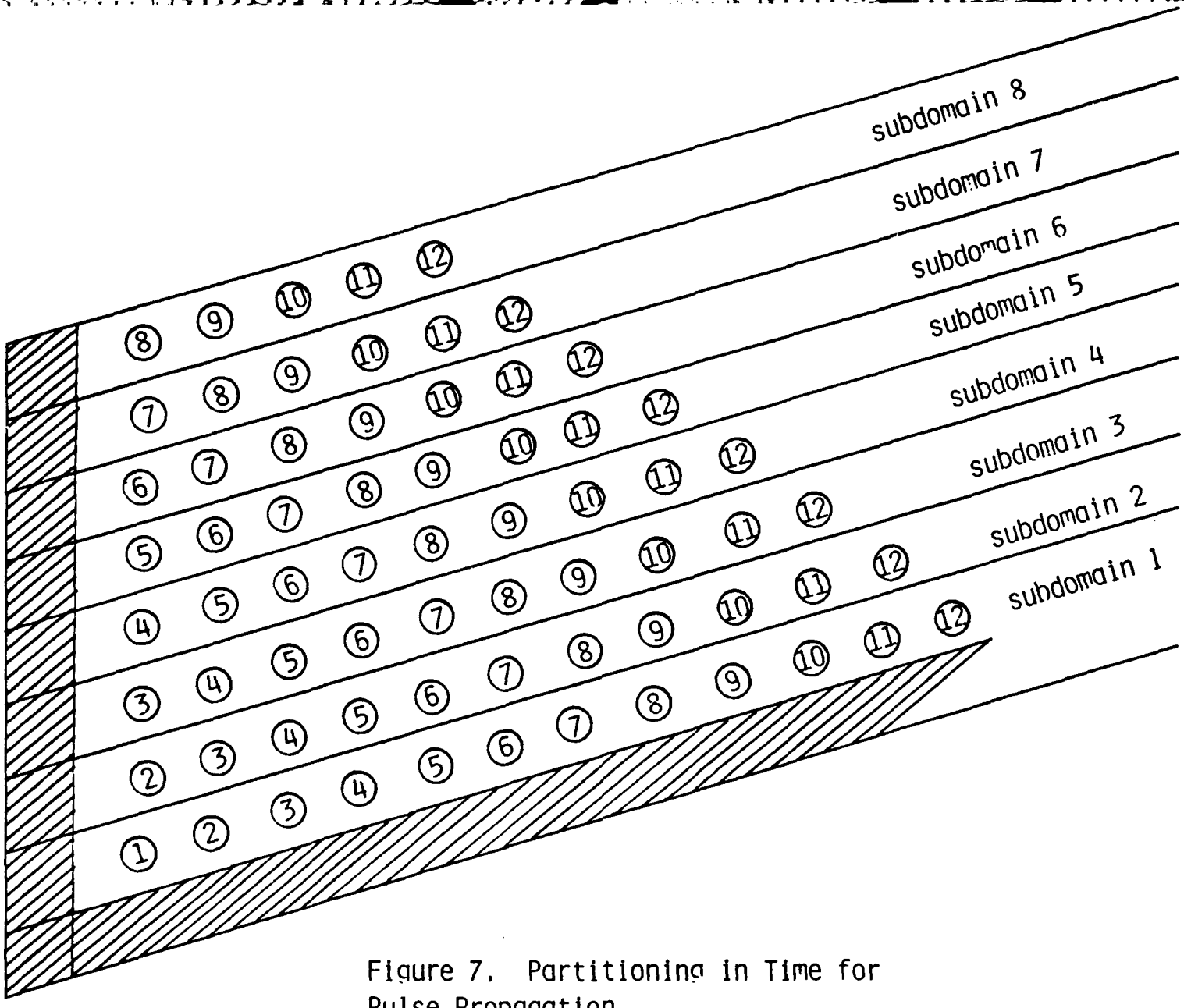


Figure 7. Partitioning in Time for Pulse Propagation

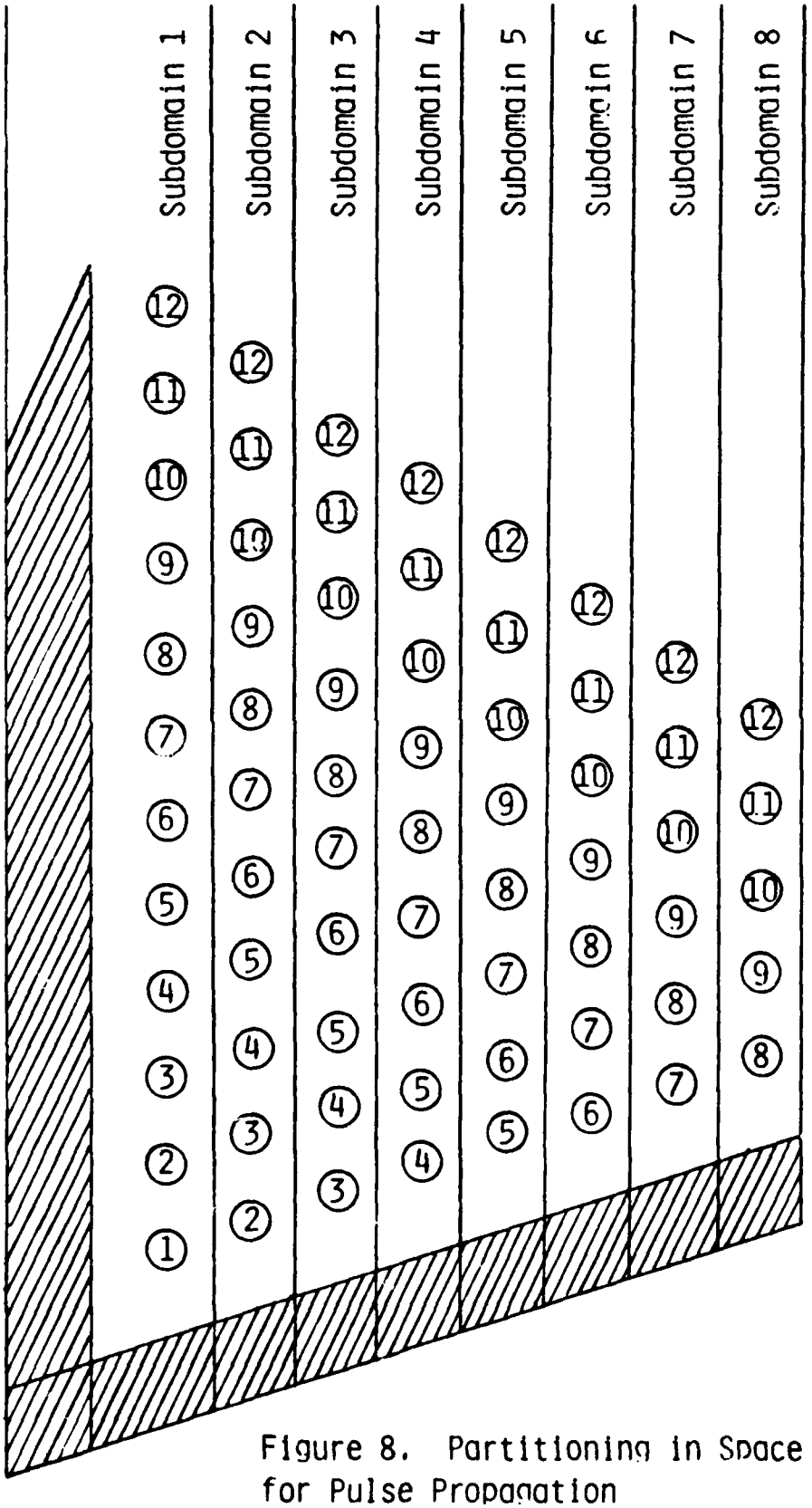


Figure 8. Partitioning in Space for Pulse Propagation

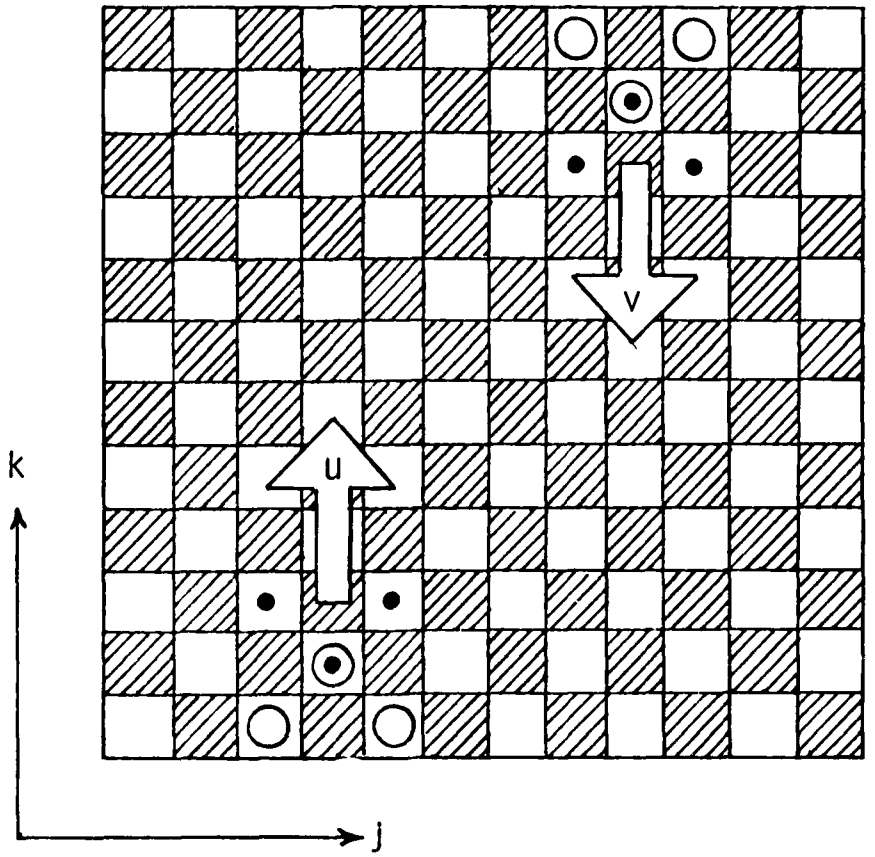


Figure 9. ADEP Finite Difference Scheme for the Two Dimensional Diffusion Problem. Solid Circles Are the Points used at the Old Time Level, Open Circles at the New Time Level. Arrows Indicate the Order of Computation.

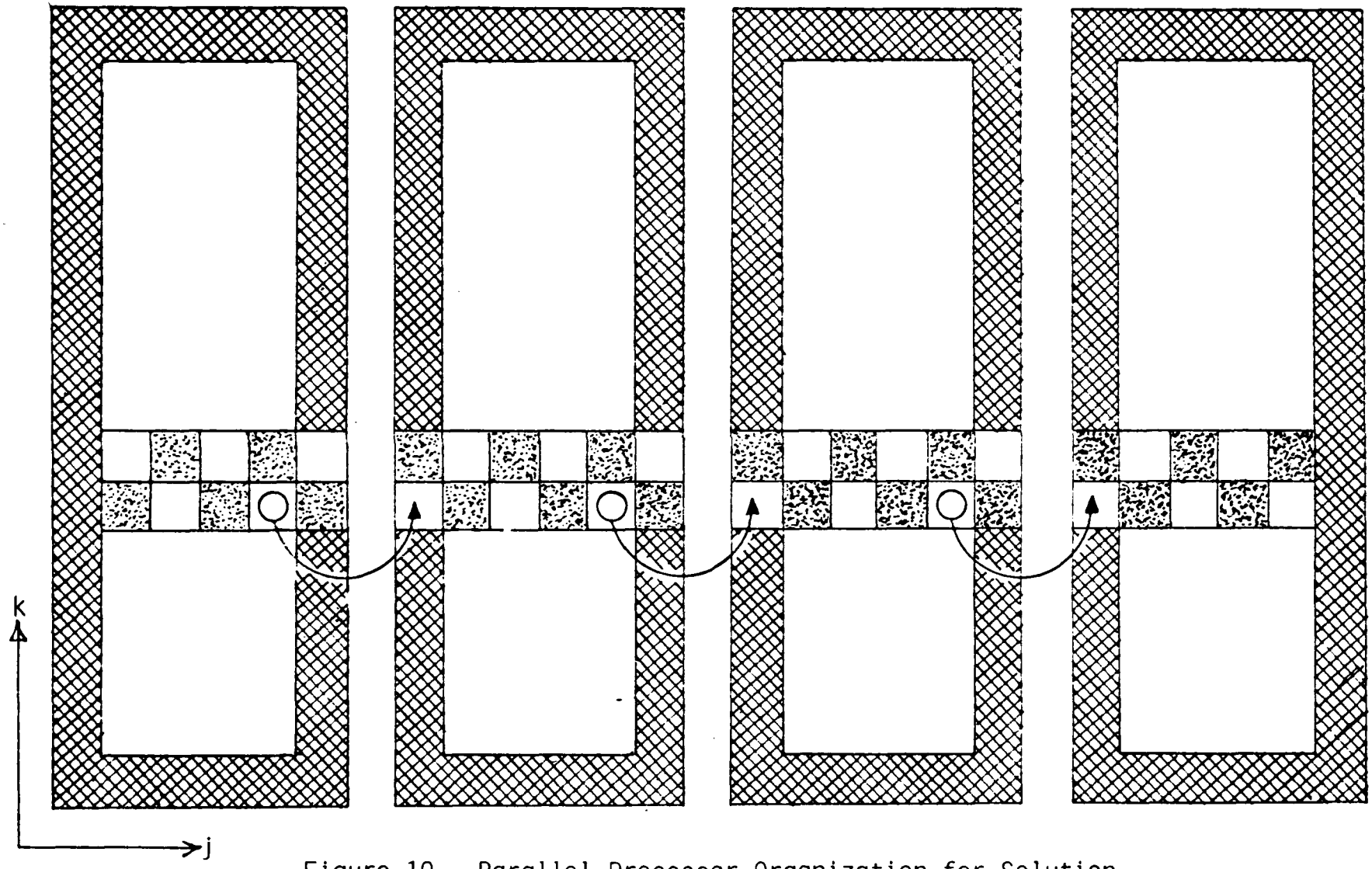


Figure 10. Parallel Processor Organization for Solution of the Two Dimensional Diffusion Equation by the ADEP Method. Data Movement is Shown for a Single Pow Only.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Manuscript: A Paradigm for the Design of Parallel Algorithms with Applications		5. TYPE OF REPORT & PERIOD COVERED Final: 1/1/83 - 12/31/83 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) I. V. Ramakrishnan and J. C. Browne		8. CONTRACT OR GRANT NUMBER(s) AFOSR F49620-83-C-0049
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Department The University of Texas at Austin Austin, Texas 78712		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Capt. A. L. Bellamy AFOSR/NM Bolling AFB, DC 20332		12. REPORT DATE December 1985 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)*		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES appeared in IEEE Transactions on Software Engineering 9, 1983, pp. 411-415.		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)* data structures, parallel algorithms, set processing algorithms, sorting and merging		

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This paper proposes a model or paradigm for the development of parallel algorithms, gives an example of the proposed paradigm, and displays algorithms developed by application of the technique. The algorithm for the merge of two ordered lists developed through application of this technique is thought to be original. The paradigm proposed is to create composite unit operations which combine data movement between data structures with a conventional operation such as compare or add. The composite operation constructed for this study is based upon partitioning the data elements into two linear lists. Exchange of data between

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Block 20: ABSTRACT (Continued)

adjacent elements in each list are then combined with compares and adds to complete the composite operations. This composite operation can be implemented on at least the following computation architectures: 1) SIMD with all processors sharing a common memory; 2) SIMD with local memories and a linear interconnection (circular pipeline or ring network of processors); 3) vector processor with common memory, such as a CDC CYBER 205 or a Cray Research CRAY-1. The algorithms developed all have the property of linear speed-up with the number of processing elements. The algorithms developed include sorting, merging, selection among sets, set interconnection, set difference, subset testing, and string matching.

A Paradigm for the Design of Parallel Algorithms with Applications

I. V. RAMAKRISHNAN AND JAMES C. BROWNE

Abstract—This paper proposes a model or paradigm for the development of parallel algorithms, gives an example of the proposed paradigm, and displays algorithms developed by application of the technique. The algorithm for the merge of two ordered lists developed through application of this technique is thought to be original. The paradigm proposed is to create composite unit operations which combine data movement between data structures with a conventional operation such as compare or add. The composite operation constructed for this study is based upon partitioning the data elements into two linear lists. Exchange of data between adjacent elements in each list are then combined with compares and adds to complete the composite operations. This composite operation can be implemented on at least the following computational architectures.

1) SIMD with all processors sharing a common memory.

2) SIMD with local memories and a linear interconnection (circular pipeline or ring network of processors).

3) Vector processor with common memory, such as a CDC CYBER 205 or a Cray Research CRAY-1.

The algorithms developed all have the property of linear speed-up with the number of processing elements. The algorithms developed include sorting, merging, selection among sets, set interconnection, set difference, subset testing, and string matching.

Index Terms—Data structures, parallel algorithms, set processing algorithms, sorting and merging.

Manuscript received October 16, 1981. This work was supported by the IBM Corporation and by the U.S. Air Force of Scientific Research under Grant AFOSR-82-0091.

The authors are with the Department of Computer Science, University of Texas at Austin, Austin, TX 78712.

I. Introduction

THERE is substantial theoretical and practical interest in the development of parallel algorithms for the common processes of numeric computing and data processing [3]-[5], [9]. There is, accordingly, a need for models and paradigms for the development of parallel algorithms akin to the traditional general techniques which have been used for sequential algorithms [10].

Lint and Agarwala [11] survey and analyze the models used in formation of parallel algorithms. All of the models for synchronous parallel algorithms are based on direct abstraction of the physical structure of the system such as common shared memory or specification of an interconnection network: "sorting on a mesh-connected computer" or "Bitonic Sort on Illiac IV," etc. This focus on a specific architecture does not generate

0098-5589/83/0700-0411$01.00

insight into the logical structure of the algorithm itself and makes transfer to other architectures difficult.

Models for asynchronous parallelism are typically very abstract: directed graphs, Petri nets, etc. This level of abstraction, while extremely powerful for structuring process interactions, often hides the actual computations of an algorithm and makes meaningful analysis of execution times difficult.

There is a need for models of parallel processing which are abstract enough to be generalizable to several architectures, but which are sufficiently descriptive to allow ready determination of execution costs. Sequential algorithms are typically modeled as operations on data structures. This model is inadequate for parallel processing because it does not consider interprocess communication.

Guidance for the development of a paradigm may be obtained from consideration of the characteristics of models of parallel processing. Models of parallel processing must include costs of synchronization and interprocess communication as well as operation counts for the computational steps of the algorithm. The paradigm proposed here is to create composite unit operations which combine data movement between data structures with conventional logical or arithmetic operations such as compare or add. The composite operation is applied to data values specified by their positions in a data structure. This paper defines and applies one example of this paradigm for the development of parallel algorithms. The composite operation is based upon partitioning the data elements into two linear lists. The composite operations are defined by combining add and compare operations with a data movement operation of value exchange between adjacent elements of each list and identically indexed elements in the separate lists. The subsequent section defines and characterizes the algorithm and its cost of execution. The analysis of execution cost is greatly simplified by the use of the composite operation since data movement and synchronization costs are included in the operations count analysis. Algorithms derived with the paradigm are given in Section III. The characteristics and architectural requirements of the algorithm are given in Section IV. The paradigm will be contrasted to the more usual approach of algorithm formulation for specific parallel architectures. Definition of data movement as occurring between data structures allows for mapping to a spectrum of parallel architectures or interconnection structures.

II. Description of the Algorithm

The data structures used are linear lists. Linear lists A and B contain the data elements upon which the algorithm will be executed and between which data movement will be defined. It will be necessary to occasionally employ auxiliary linear lists which we will note by C, FLAG, etc. These auxiliary variables will be used to hold intermediate results for some algorithms.

We first write the algorithm in the simple case where the number of elements in each data array is n and the number of processors is n. We then give the algorithm in general form where A and B may be of different dimension and where the number of processors k may be less than the dimensions of A or B. Proofs of linear speed-up and bounds for computation time are then given for this general case.

A conventional notation is used. The computational steps within **cobegin-coend** brackets are processed in parallel with the degree of parallelism specified by the range of the index of the **cobegin**. For example, the notation

```
cobegin: i <0:n-1>
   operation (A[i], B[i])
coend
```

implies n parallel executions of **operation** $(A[i], B[i])$ on each of the elements $<0 \leq i \leq n-1>$ of A and B. Processors are assigned a number from 0 to $n-1$ (or from 0 to $k-1$) and the processor of index i acts upon the data elements of index i.

Algorithm (for n processors and n elements in each linear lists):

```
begin
   for i = 0 to n-1 do
      cobegin: j <0:n-1>
         operation (A[j], B[j], C[j])
      coend
end.
```

Each execution of the **cobegin-coend** bracket executes simultaneously the as yet undefined composite arithmetic/logical plus data movement operation on the three tuple $A[j], B[j], C[j]$. This is then repeated n times to complete the execution of the algorithm.

In order to conveniently display that linear speed-up in the number of processors is obtained, we write the algorithm where A and B are of size n and m, respectively, and k processors are used. We restrict the number of processors k to the situation where $m = ck$ and c is integer.

Algorithm (k processors, $m = ck$, $m<n$):

```
begin
   for i = 0 to n-1 do
      for j = 0 to m-1 in steps of k do
         cobegin: l <0:k-1>
            operation (A[l], B[l], C[l])
         coend
end.
```

All algorithms expressible in this form show speed-up proportional to the number of processors if we neglect load time and output time, as is commonly the case at this level of resolution of detail.

Lemma 1: Using k processors the algorithm uses $0(mn/k)$ steps.

Proof: Let $m=ck$ for some integer c. The inner loop takes $0(m/k)$ steps. Therefore the outer loop takes $0(mn/k)$ steps.

Definition: Let $|A|$ and $|B|$ denote the sizes of the arrays A and B, respectively. Let $\min(|A|, |B|)$ and $\max(|A|, |B|)$ denote the minimum and maximum of the sizes of the arrays A and B, respectively.

Theorem 1: Using $\min(|A|, |B|)$ processors the algorithm uses $\max(|A|, |B|)$ steps.

Proof: $m=|B|$, $n=|A|$ and $m < n$. Therefore, $m=\min(|A|, |B|)$ and $n=\max(|A|, |B|)$. Substituting $k=m$ in Lemma 1, the theorem follows.

III. Application

The utility of the technique is illustrated by developing algorithms for sorting, merging, selection, set-problems, and string-matching. In all the algorithms we use $\min(|A|, |B|)$ processors in order to simplify the presentation. Extension to the case where the number of processors, k, is less than $\min(|A|, |B|)$ is straightforward so long as $m = ck$ and c is integer.

A. Sorting

Sorting of a sequence of elements can be performed by ranking each element in the sequence. Rank of an element x is defined to be the number of elements greater (less) than x. Ranking is done by comparing every element against every other element in the sequence. Let A and B be two arrays each of size n, each containing the list of elements to be sorted. Initially $C[i] = 0$ for $i = 0, 1, 2 \cdots$.

Sorting Algorithm:

```
begin
  for i=0 to n-1 do
    cobegin: j<0:n-1>
      if A[j] > B[j] then
          C[j] := C[j] +1;
          A[j] = A[(j+i) mod n]
    coend
end.
```

≡ operation $(A[j], B[j], C[j])$

The "←" is a data transfer between the two list positions. The composite operation here consists of comparison between two identically indexed elements of A and B, incrementing a rank counter C and circular left-shift of the elements of A. This sorting algorithm correctly ranks all the elements in the array B. The rank of identical elements will, however, be the same. Consequently, completion of the sort by moving the data elements to their positions by rank will result in all identical elements being moved to the same locations in array A. In order to avoid this, we adopt the following solution. Identical elements are placed in consecutive locations in the final array on the basis of their relative positions in the array B. If B_i and B_j are two identical elements, B_j follows B_i in the sorted sequence if $i<j$ and vice versa if $i>j$. An element in the jth position in the array B is compared against elements to its left after $n-j$ iterations of the for loop. Processor P_j which always compares the j element in array B with every other element, updates its count on comparing with an equal element only after $n-j$ iterations.

Sorting Algorithm (identical list elements):

```
begin
  for i=0 to n-1 do
    cobegin: j <0:n-1>
      if (A[j] > B[j])
    or ((A[j] = B[j])
        and j > (n-i))
          then
        C[j] := C[j] + 1;
        A[j] = A[(j+i) mod n]
    coend
end.
```

B. Selection

Given a sequence of elements $a_1, a_2, \cdots, a_n$, selection involves choosing the kth largest (smallest) element in the sequence. By ranking the elements, selection can be performed by indexing. The technique used to rank elements in the sorting algorithm is used for selection.

C. Merging

Array A is of size n and B is of size m. Initially $A[1] \leqslant A[2] \leqslant \cdots A[n]$ and $B[1] \leqslant B[2] \cdots \leqslant B[m]$.

Merging Algorithm:

```
begin
  for i=0 to n-1 do
    cobegin: i<0:m-1>
      if A[j] > B[j] then
          A[j] ↔ B[j] then
          A[j] = A[(j+i) mod n]
    coend
end.
```

The "↔" operator interchanges the elements of two lists.

On termination, $A[1] \leqslant A[2] \leqslant \cdots \leqslant A[n] \leqslant B[1] \leqslant \cdots \leqslant B[m]$. This algorithm has not been previously found in the literature and is believed to be a new algorithm for parallel merger of two lists. Proof of this algorithm is given in [7].

D. Set Intersection

The two sets of elements are contained in arrays A and B. Array A is of size n and array B of size m ($n \geqslant m$). Let FLAG be a Boolean array of size m. Initially, all the values in the Boolean array are false.

Set-Intersection Algorithm:

```
begin
  for i=0 to n-1 do
    cobegin: i <0:m-1>
      if A[j] = B[j] then
        FLAG[j] ← true;
        A[j+1] ← A[j] mod n
        A[j] = A[(j+i) mod n]
    coend
end.
```

On termination the true values in the FLAG array correspond to elements in the array B that are also contained in array A.

E. Set Difference

Arrays A and B represent the elements in the two sets as in Section III-D. AFLAG and BFLAG are two Boolean arrays. AFLAG is of size n and BFLAG is of size m ($n \geqslant m$). Initially, the values in both the boolean arrays are false.

Set-Difference Algorithm:

```
begin
  for i=0 to n-1 do
    cobegin: j <0:m-1>
      if A[j] = B[j] then
```

```
            begin AFLAG[j] ← true;
                  BFLAG[j] ← true;
                  A[j] = A[(j+i) mod n]
            end
      coend
end.
```

On termination, the false values in AFLAG correspond to elements in array A that are not in array B, and the false values in BFLAG correspond to elements in array B that are not in array A.

F. Subset Testing

Arrays A and B denote sets of elements as in Section IV-D and III-D. FLAG is a boolean array of size m. Initially, all the values in C are false.

Subset Testing Algorithm:

```
begin
   for i=0 to n-1 do
      cobegin: j <0:m-1>
         if A[j] = B[j] then
            FLAG[j] ← true;
         A[j] = A[(j+i) mod n]
      coend
end.
```

On termination, if all the values in FLAG are true, then B is contained in A.

G. String-Matching

List A holds the text string and List B holds the pattern string. String-matching involves determining whether the pattern held in B occurs as a substring of the text held in A.

The subset-testing algorithms of Section III-F can be used for string-matching with a slight modification. Instead of testing the FLAG array at the termination of the algorithm, the array is tested after every iteration. If after any iteration the value of all elements in the FLAG array are true, then the pattern held in B occurs as a substring in A.

IV. Analysis of the Proposed Paradigm

The usual procedure for establishment of parallel algorithms is to map a sequential algorithm to some specific parallel architecture. The execution cost is determined by separately counting operations and data movement costs. The technique of defining a composite operation against a data structure both generalizes this approach and simplifies execution cost analysis.

The composite operation in these examples uses only the adjacent elements in each list and identically indexed elements between the two lists. This composite operation can be realized in many commonly considered models of parallel architectures, including SIMD with common memory, a circular pipeline of SIMD processors with no common memory, and a vector instruction architecture. Realization on SIMD with common memory and with vector instructions is obvious. Realization on a circular pipeline can be realized as shown in the schematic of Fig. 1.

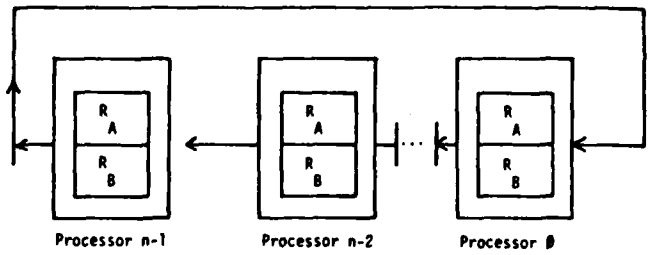


Fig. 1. Circular pipeline of processors architecture.

R_A and R_B are registers to hold the values of lists A and B. The registers of each processor can compare and exchange values. The processors are connected by one-way pipes. A "shift" instruction moves the values from R_A of processor i to R_A of processor $i+1$ with end-around for $n-1$ to 0. This basic structure can be augmented with additional registers for each processor as needed for FLAG, etc., vectors. This simple and regular architecture suggests VLSI implementation. It should be noted that this architecture will not implement the string matching algorithm without special provision for the AND of the Boolean values in the FLAG vector at each cycle.

A wide spectrum of algorithms can be constructed by composing only compare and add with the very simple data movements among so simple a data structure as a linear list. Execution costs, including data movement, are readily determined for a number of different architectures for each algorithm. The operations and algorithms also suggest effective architectures for given problems. Definition of composite operations on square arrays leads to formulation of graph and image processing algorithms. These will be explained in a later paper.

V. Summary

We have proposed and described a paradigm for the design of parallel algorithms. We have applied the technique to produce a number of algorithms. The algorithms produced by the application of the paradigm have desirable properties such as linear speed-up and readily obtainable bounds. The technique produces algorithms which are applicable to a wide variety of computational architectures. We propose that the paradigm of specifying composite operations against data structures is a basis for at least one direction of study of parallel algorithms.

Acknowledgment

The authors gratefully acknowledge the suggestions of Dr. M. Gouda and Dr. D. Fussell for improving the clarity of the paper.

References

[1] M. J. Flynn, "Very high speed computing systems," *Proc. IEEE*, vol. 54, pp. 1901-1909, Dec. 1976.

[2] D. S. Hirschberg, "Fast parallel sorting algorithms," *Commun. Ass. Comput. Mach.*, vol. 21, no. 8, pp. 657-661, Aug. 1978.

[3] H. T. Kung, "Let's design algorithms for VLSI systems," CMU-CS-79-151. Also presented at the Conf. Very Large Scale Integration: Arch. Des. Fabrication held at Calif. Inst. Technol., Jan. 22-24, 1979.

[4] —, "The structure of parallel algorithms," CMU-CS-79-143. Also to appear in *Advances in Computers*, vol. 19. New York: Academic.

[5] K. Levitt and W. Kautz, "Cellular arrays for the solution of graph problems," *Commun. Ass. Comput. Mach.*, vol. 15, no. 9, pp. 789–801, Sept. 1972.
[6] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1979.
[7] I. V. Ramakrishnan and J. C. Browne, "Merging on cyclically-interconnected processors," Tech. Rep., Dep. Comput. Sci., Univ. Texas at Austin, May 1980.
[8] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer," *Commun. Ass. Comput. Mach.*, vol. 20, no. 4, pp. 263–271, Apr. 1977.
[9] F. L. Van Scoy, "The parallel recognition of classes of graphs," *IEEE Trans. Comput.*, vol. C-29, pp. 563–570, July 1980.
[10] S. Sahni and E. Horowitz, *Fundamentals of Computer Algorithms*. Baltimore, MD: Comput. Sci. Press, 1978.
[11] B. S. Lint and T. Agarwala, "Communications issues in the design and analysis of parallel algorithms," *IEEE Trans. Software Eng.*, vol. SE-7, pp. 174–188, 1981.

I. V. Ramakrishnan received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, in 1975, and the M.E. degree in automation from the Indian Institute of Science, Bangalore, in 1977.

From 1977 to 1979 he was a Scientific Officier at the National Center for Software Development and Computing Techniques (NCSDCT), Tata Institute of Fundamental Research, Bombay, India, where he was involved in the design and implementation of a capability-based operating system for a closely coupled network of multiprocessors. His current research interests are in VLSI algorithms. He is currently with the Department of Computer Science, University of Texas at Austin.

James C. Browne was born in Conway, AR, on January 16, 1935. He received the B.A. degree from Hendrix College, Conway, AR, in 1956, and the Ph.D. degree in physical chemistry from the University of Texas at Austin in 1960.

He worked from 1960 to 1964 as an Assistant Professor of Physics at the University of Texas at Austin. From 1964 to 1965, he was at the Queen's University of Belfast, Belfast, Northern Ireland, on a National Science Foundation Postdoctoral Fellowship. From 1965 to 1968, he was Professor of Computer Science and Director of the Computation Laboratory at the Queen's University of Belfast. In 1968 he rejoined the University of Texas as Professor of Computer Sciences and Physics, and served as Chairman of the Department of Computer Sciences 1968–1969 and from September 1971 until January 1975. His research interests include operating systems, database systems, performance evaluation, and parallel computing. He has published about 125 refereed papers on technical subjects.

Dr. Browne is a Fellow of the American Physical Society and the British Computer Society, and a member of the Association for Computing Machinery.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Manuscript: A Comparison of Circuit Switching and Packet Switching for Data Transfer in Two Simple Image Processing Algorithms	5. TYPE OF REPORT & PERIOD COVERED Final: 1/1/83 - 12/31/83 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) M. Yasrebi, S. Deshpande and J. C. Browne	8. CONTRACT OR GRANT NUMBER(s) AFOSR F49620-83-C-0049
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Department The University of Texas at Austin Austin, Texas 78712	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Capt. A. L. Bellamy AFOSR/NM Bolling AFB, DC 20332	12. REPORT DATE December 1985 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

appeared in Proceedings of the 1983 International Conference on Parallel Processing.

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The communication costs for parallel versions of two simple algorithms used in image processing are compared in packet switching and circuit switching formulations. The two algorithms are smoothing and histogramming. The histogramming algorithm, the recursive doubling algorithm of Stone, is studied over a range of processor numbers and pixel intensity resolution. The packet and circuit switching properties of the interconnection networks of the multiprocessor systems are based on two network architectured multiprocessors which are well-documented in the literature, PASM and TRAC. Communication based upon

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Block 20: ABSTRACT (Continued)

circuit switching generally gives a somewhat lower communication cost with the advantage increasing with pixel intensity resolution. The results of the analysis suggest a high utility value for including both circuit switching and packet switching functionality in the networks of network architectured multiprocessor systems.

A COMPARISON OF CIRCUIT SWITCHING AND PACKET SWITCHING FOR DATA TRANSFER IN TWO SIMPLE IMAGE PROCESSING ALGORITHMS

by

Mehrad Yasrebi
Communication Products Division
IBM Corporation
Research Triangle Park, NC
and
Sanjay Deshpande and J.C. Browne
Department of Computer Sciences
The University of Texas at Austin

Abstract

The communication costs for parallel versions of two simple algorithms used in image processing are compared in packet switching and circuit switching formulations. The two algorithms are smoothing and histogramming. The histogramming algorithm, the recursive doubling algorithm of Stone, is studied over a range of processor numbers and pixel intensity resolution. The packet and circuit switching properties of the interconnection networks of the multiprocessor systems are based on two network architectured multiprocessors which are well-documented in the literature, PASM and TRAC. Communication based upon circuit switching generally gives a somewhat lower communication cost with the advantage increasing with pixel intensity resolution. The results of the analysis suggest a high utility value for including both circuit switching and packet switching functionality in the networks of network architectured multiprocessor systems.

Introduction and Overview

This paper compares the communication costs for executing two algorithms used in image processing on three parallel computer architectures. The purpose of the comparison is to evaluate packet switching and circuit switching modes of data movement for interconnection network based multiprocessors. The two algorithms used for the comparison are computation of histograms of the intensity values of pixels of an image and smoothing of gray level data across the pixels of an image.

The model for a packet switching architecture is the Partitionable SIMD/MIMD (PASM) System for Image Processing and Pattern Recognition [Siegel81]. The model for a circuit switching architecture is the Texas Reconfigurable Array Computer (TRAC) [Sejnowski80]. The third architecture, all processors sharing a common bus [Bhuyan82], is given as a baseline for the comparison. An analysis of communication costs for the two algorithms executing on PASM has been given in [Siegel81]. The results of an analysis of the execution of the two algorithms in a circuit switching formulation based on TRAC are given here. Space limitations preclude detailing of the analysis.

Communication Analysis for Parallel Algorithms

The major factors determining communication cost for the execution of parallel algorithms on interconnection network (ICN) based multiprocessors include: (i) the topology of the ICN and the configuration of resources on the ICN, (ii) the mapping of the data movement requirements of the algorithm upon the ICN, (iii) choice of switching methodology, (iv) the latency and bandwidth properties of the ICN, and (v) the unit sizes and the total volume of the data to be moved. This paper focuses on the impact of switching methodology and data volume on communication cost.

The choice of packet switching or circuit switching as the mode of network data path establishment can have a substantial effect on each of these architectural parameters. Packet switching tends to give flexibility in topology but fixed unit transfer sizes. Circuit switching tends toward less flexibility in topology, greater flexibility in unit size for transfers, but a longer transfer latency time. Packet switching may also introduce bandwidth degradation due to path contention while circuit switching may introduce path blockages which limit realizable network topologies for all networks short of full cross-bars.

The measure of communication cost is elapsed time. The communication times given herein are reported as number of memory cycles. We assume, in order to normalize computation costs across architectures, that an integer addition takes a single memory cycle and that updating a histogram vector element requires two integer additions. The speed-up of a multiprocessor over a uniprocessor is the ratio of total execution times, T_E, where $T_E = T_{COMM} + T_{COMP}$. All LOG's in this paper are in base 2 unless otherwise noted. The data paths of each ICN are taken to be one integer word in width. For the multistage ICN's of PASM and TRAC it is assumed that a unit of data moves through one stage of the ICN on each memory cycle.

Definition of Architectures

Communication cost for execution of the two algorithms is compared for three ICN-based multiprocessor architectures. The single shared bus architecture (Figure 1) has been characterized by Bhuyan and Agrawal [Bhuyan82]. It is a baseline for ICN-based multiprocessors. There is no distinction between packet and circuit switching in this model of communication. The model for a packet switching data movement architecture is PASM [Siegel81]. The ICN of PASM connects complete processing elements as shown in Figure 2.

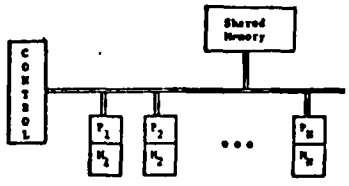


Fig. 1. A Multiprocessor with a Shared Bus

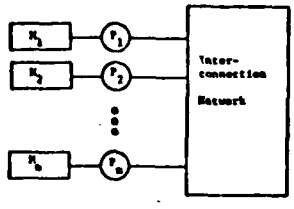


Fig. 2. PE-to-PE Configuration

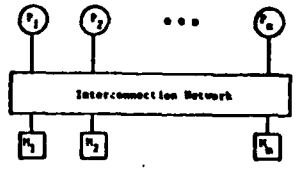


Fig. 3. Processor-to-Memory Configuration

The interconnection networks proposed for PASM are the generalized cube and the augmented data manipulator (ADM) [Siegel79]. These two networks are optimal for histogramming in the sense that all permutations for the algorithm can be realized by both networks in a single pass. Thus packet transfers can take place without blocking.

The model for a circuit switching data movement architecture is TRAC [Sejnowski80]. TRAC places processors at the apex nodes and memories at the base nodes of its ICN (Figure 3). The ICN of TRAC is an SW-Banyan [Premkumar80] with nodes having spread of two and fanout of three for its ICN. Processor configurations are formed by establishing circuits in the ICN joining processors to memory units. Data flow between processors for different stages of the algorithms can be realized by dynamically switching memories between processor-memory configurations. This network also implements trees of circuits joining one memory to many processors in which any one circuit may be activated and/or deactivated by a single processor instruction. These tree circuits are called shared or switchable memory trees. Data flow between processors may be implemented using this capability by a sequence of circuit activations and deactivations (among the circuits following the tree).

The ICN of TRAC actually implements both circuit switching and packet switching but only the circuit switching mode of use is modeled in the equations given following.

The Algorithms and Their Mapping to the Architecture

Histogramming and smoothing are among the basic operations of image processing, although not usually rate determining steps in the computations. Attention to detailed parallel formulations of major computational steps of image processing such as thresholding and edge detection is needed. It is assumed in the analysis following that the picture is M*M pixels in size ($M=2^m$) and that N ($N=2^n$) processors are available. The resolution of each pixel is λ bits.

Histogramming Algorithm

The parallel algorithm for histogramming is the recursive doubling algorithm of Stone [Stone75]. The structure of the algorithm is shown in Figure 4 for N=8. N partial histograms are computed in parallel at level 0. Each partial histogram is a vector of length 2^λ. The partial histograms are then added in pairs in parallel for LOG(N) stages to complete the algorithm.

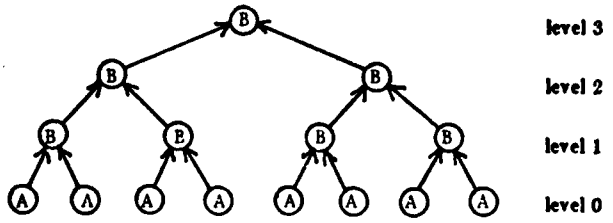


Figure 4: Recursive Doubling Algorithm for Histogramming

Partial histograms are shown at level 0 by A's and vector additions by B's. $N/2^i$ transfers of vectors are done between level (i-1) and level i. The computation time, T_{COMP}, for this algorithm under the assumptions made here is proportional to $T_{COMP} = M^2/N + 2^\lambda \text{ LOG}(N)$.

A Packet Switching Formulation of Recursive Doubling Histogramming - Siegel *et al* [Siegel81] have given a thorough analysis for the execution of this algorithm on PASM. We adopt the results of this study as our packet switching model of recursive doubling histogramming. It is commonly the case that further steps in the analysis of the image require thresholding so that the final histogram vector must be collected in one processor and the threshold value distributed. The total communication time for this formulation of the algorithm is

$$T_{COMM} = [(\text{LOG}(N) + 2^\lambda) + 2] \times \text{LOG}(N).$$

- $\text{LOG}(N)$: travel time through the ICN
- 2^λ: switch setting time
- 2: number of levels in the ICN

A Circuit Switch Formulation of Recursive Doubling Histogramming Based on Tree Circuits - Figure 5 illustrates the structure of the circuit switched data movement formulation of recursive doubling for an 8 processor-8 memory configuration.

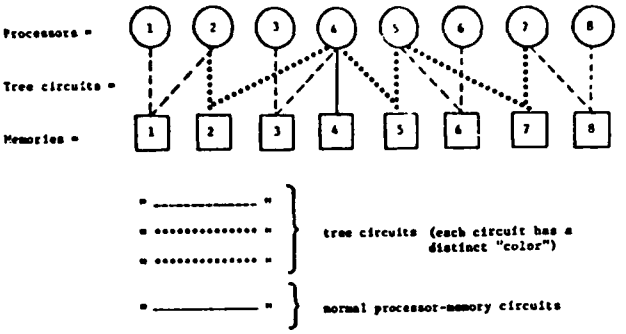


Figure 5: Circuit Switching Using the Tree Circuit Formulation

The M^2 pixels are evenly partitioned among the 8 memories. Each processor computes a partial histogram vector and stores it in the corresponding memory. The computation is then completed in LOG(8)=3 stages of adding partial vectors with the full histogram computed by processor 3 and stored in memory 5. The tree circuits of Figure 5 implement the successive communication paths between levels in Figure 5. The "——" tree circuits implement the data flow between levels 0 and 1 in Figure 4, "ooooo" the data flow between levels 1 and 2 and "....." the data flow between levels 2 and 3. There is a regular pattern of using first the verticals and then the diagonals of each type of tree circuit. Each tree circuit type has a unique number (called COLOR in correspondence with graph theory). LOG(N) colors are required to implement the algorithm in this formulation. Path selection (activation and deactivation) in all tree circuits of identical COLOR can be done in parallel with a latency time proportional to LOG(N)/2. The ICN of TRAC can implement the tree circuit pattern of Figure 5 without blockage. The total communication cost for this formulation of recursive doubling histogramming is

$$T_{COMM} = [\text{LOG}(N)]^{-1} \sum_{i=1}^{\text{LOG}(N)} N/2^i \; [\text{LOG}(N) + 2] + \text{LOG}(N)$$
$$= (3/2)(N-1)$$

- $(3/2)(N-1)$: time to switch all memory with identical COLOR
- $\text{LOG}(N)$: latency time

A Circuit Switching Formulation of Recursive Doubling Based on Direct Reconfiguration - Another formulation based on circuit switching can be developed by directly reconfiguring the ICN after each step (level in Figure 4) of the algorithm to conform to the data movement path required at each stage of the algorithm. Each configuration step involving establishment of a circuit between a given processor and a set of memories must be done serially. Thus use of the tree-circuit based algorithm is faster by a factor of LOG(K) where K is the number of COLORs available.

The Smoothing Algorithm

Smoothing is replacement of the intensity of each pixel by the mean of the intensity of the given pixel and its nearest neighbors.

Packet Switching Formulation of Smoothing - Siegel *et al* [Siegel81] have formulated and analyzed a packet data movement formulation of the smoothing algorithm. They show a speed-up of

about .8N for a 1024 processor configuration. This estimate is extremely conservatively based. A greater speed-up is probable.

Circuit Switching Formulations of Smoothing - A circuit switching structure for the smoothing operation is suggested by the fact that each computation requires only nearest neighbors. Therefore if the pixels are stored by columns then a processor will need simultaneous access to three columns (say k-1,k,k+1) to execute the computations on column k. A realization of this representation of the smoothing algorithm is given in Figure 6. Extra zero valued rows and columns of pixel values are added to each formulation of boundary conditions. The solid lines of Figure 6 are normal circuits. The dotted lines are tree circuits from which leaf-root paths can be selected. Processor 1 computes in sequence the smoothed values for the pixels in columns 1, 2 and 3. Processor 2 will simultaneously and in sequence compute the smoothed values for the pixels in columns 4, 5 and 6. P1 and P2 must share access to pixel columns 3 and 4. The execution procedure described preceding allows this sharing to be accomplished without conflict if the required circuits can be established in the network. This two processor configuration obviously extends to an N processor 3N-memory configuration so long as the memory unit can hold an entire column of pixel values. A TRAC-like ICN can realize these configurations so long as these restrictions are met. It is also the case that the necessary data movement can be realized by reconfiguration of normal circuits. This is not the method of choice so long as the conditions for a tree circuit representation can be met.

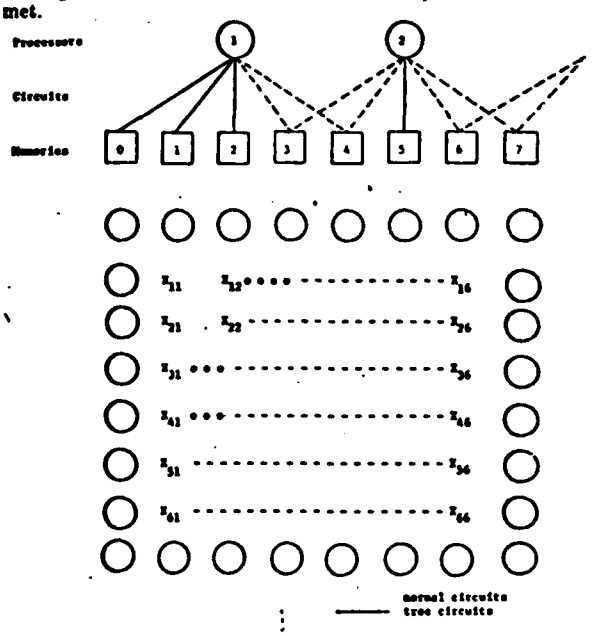


Figure 6 A Storage Structure and Circuit Configuration for Parallel Smoothing

It may be desired to use a degree of parallelism greater than M (N>M). Then the columns of pixels must be decomposed into vectors of length M/k where N=kM. In this case the establishment of circuits is dependent upon k and may not always be possible. A formulation using both circuit switching and packet routing capabilities for TRAC has been worked out. The pixels appearing at the boundaries created by partitioning of columns have their "nearest neighbors" sent to them by packet movement. This "mixed" communication mechanism is still of lower cost than a pure packet based mechanism. The case N<M (for $N=2^i$, $m=2^j$) is handled by assigning multiple (2^k) columns to processors. This case raises no new problems.

We give here numbers only for the circuit switching representation where N=M and data movement is via tree circuit activation and deactivation. Then the total communication cost is (N/2) LOG(N) (assuming deactivation and activation of all tree circuit paths is done in parallel). If N=M=512, then only 256*9=2304 memory cycles are required for data communication. This is negligible compared to the C*512*512 arithmetic operations on the pixels ($C \geq 10$ and probably $C > 10^2$) since indexing must be accomplished as well as the addition and division of smoothing itself.

We thus conclude that for smoothing data movement costs will be essentially trivial for both packet and circuit switching representations.

Speed-up Analysis and Discussion

Figure 7 shows the net speed-up versus the number of processors for M=1024 and λ=8. There is, in this case, little difference between formulations based on different switching strategies for moderate numbers of processors. There is the suggestion that circuit switching will yield superior performance for large numbers of processors.

Figure 8 shows the speed-up factor as a function of λ for M=1024, and N=256. The amount of data transferred grows exponentially as λ. Thus circuit switching data movement shows a strong advantage as λ increases since the cost of data movement in the circuit switching strategy given here is constant with respect to data volume until the capacity of a memory unit is exceeded.

Smoothing on the other hand shows advantage for packet switching since there are cases where a pure circuit switching formulation becomes rather complex.

The bottom line with respect to parallel histogramming is that circuit switching has an advantage resulting from flexibility in the unit size of transfers and in stability with respect to algorithm parameters but that well-designed architectures should give similar performance for small to moderate numbers of processors.

Circuit switching and packet switching are both extremely efficient for parallel smoothing. Packet switching has an advantage over circuit switching with respect to application of degrees of parallelism with N>M for parallel smoothing. This advantage arises from the greater flexibility in communication topology.

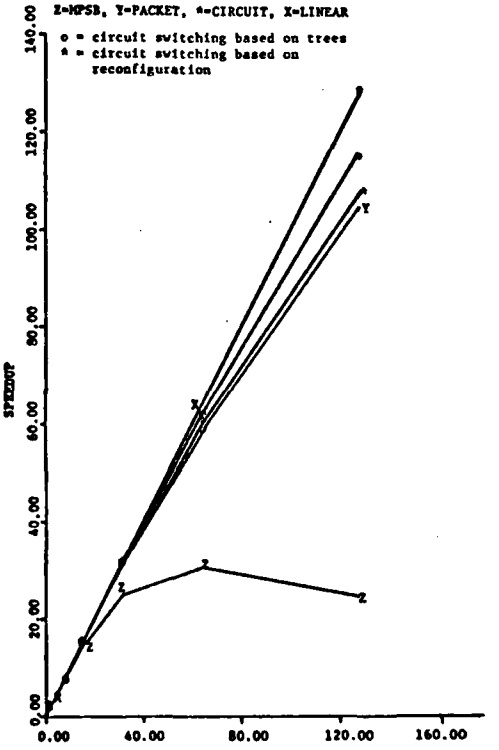


Figure 7 Speedups versus the Number of Processors (M=1024, λ=8)

There is suggestion from these two algorithms that implementation of both packet and circuit switching facilities in the ICN's for multiprocessors will give lower communication cost and greater net speedup than either used separately.

Acknowledgements

This work was sponsored by the Air Force Office of Scientific Research under Grant Number AFOSR-82-0091 and by the National Science Foundation under Grant Number MCS-8116099.

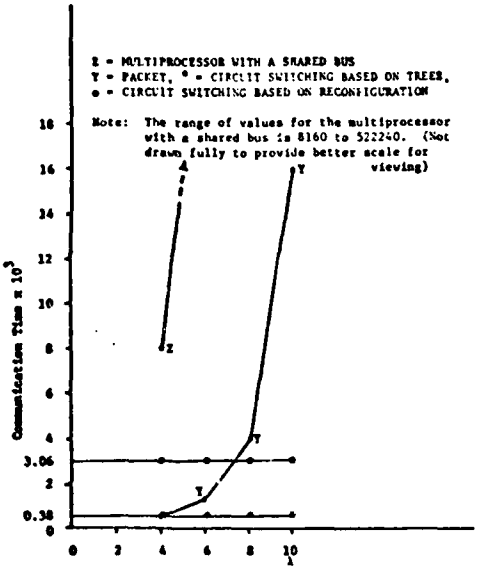


Fig. 8 Communication Time Versus λ (N=256)

References

[Bhuyan82] Bhuyan, L.N. and Agrawal, D.P., "Applications of SIMD Computers in Signal Processing", AFIPS Conf. Proc. 51, pp. 135-142, 1982.)

[Feng81] Feng, Tse-yun, "A Survey of Interconnection Networks", Computer 14(2), December 1981.

[Premkumar79] Premkumar, U.V., et al, "Interprocessor Communication on the Texas Reconfigurable Array Computer", in 1st Int. Conf. on Distributed Computer Systems, 1979.

[Premkumar80] Premkumar, U.V., et al, "Design and Implementation of the Banyan Interconnection Network in TRAC", AFIPS Conf. Proc., May 1980.

[Sejnowski80] Sejnowski, M.C., et al, "An Overview of the Texas Reconfigurable Array Computer", NCC Conf. Proc., 1980.

[Siegel79] Siegel, H.J., "Interconnection Networks for SIMD Machines", Computer 12, June 1979.

[Siegel81] Siegel, H.J., et al, "PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition", IEEETC C-30(12), December 1981.

[Stone75] Stone, H., Introduction to Computer Architectures, Science Research Associates, Inc., 1975.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Manuscript: TRAC: An Environment for Parallel Computing		5. TYPE OF REPORT & PERIOD COVERED Final: 1/1/83 - 12/31/83 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* J. C. Browne		8. CONTRACT OR GRANT NUMBER*(s)* AFOSR F49620-83-C-0049
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Department The University of Texas at Austin Austin, Texas 78712		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Capt. A. L. Bellamy AFOSR/NM Bolling AFB, DC 20332		12. REPORT DATE December 1985 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)*		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES appeared in Proceedings of COMPCON 1983		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*		

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This paper defines one set of requirements for a successful general purpose parallel architecture, describes the design concepts of the Texas Reconfigurable Array Computer (TRAC) and then demonstrates that the TRAC architecture fulfills these requirements. It will be seen that TRAC implements a general purpose parallel computation system through its ability to implement a spectrum of single purpose architectures. Special attention is paid to architectural support for software and to the I/O problems for a many-processor architecture.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

TRAC: AN ENVIRONMENT FOR PARALLEL COMPUTING

J. C. Browne

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

Abstract

This paper defines one set of requirements for a successful general purpose parallel architecture, describes the design concepts of the Texas Reconfigurable Array Computer (TRAC) and then demonstrates that the TRAC architecture fulfills these requirements. It will be seen that TRAC implements a general purpose parallel computation system through its ability to implement a spectrum of single purpose architectures. Special attention is paid to architectural support for software and to the I/O problems for a many-processor architecture.

Requirements for Effective Parallel Computing

A parallel computer architecture is founded upon a set of assumptions about the nature of the workload it is to execute and the technology available for implementation. These assumptions in turn define the requirements for the architecture. This section lists the assumptions and requirements which underlie the design of the Texas Reconfigurable Array Computer (TRAC).

1. An effective general purpose parallel architecture must be able to realize a spectrum of models of parallel computation. (See Section 2 for definition of models of parallel computation.) In many cases efficient parallel formulation of most significant problems will require multiple modes of parallel computation.

2. High performance communication between parallel processes is as important as high performance computations.

3. A successful many processor architecture must include an effective solution to the problem of distributing data from I/O.

4. Software will be the major bottleneck in the application of parallel computing.

5. A successful family of parallel architectures will be founded on a concept base which is extensible across a broad range of numbers of processors.

The validity of the arguments that TRAC can be the basis for a very high performance parallel architecture depends on the validity of this requirements analysis.

The sections of this paper which follow will show that TRAC offers one total system architecture which meets the requirements of the problem domain defined by this set of assumptions.

Models of Computation

It is useful, before going on to the discussion of TRAC design principles, to give our perspective on parallel computing. Problems, algorithms, architectures and languages can all be characterized by the model of computation which they require for execution or which they realize.

The essential element in the understanding of the development of algorithms, software and applications for parallel architectured computer systems is an understanding of the models of computation required by significant problems and realized on hardware architectures. A model of computation for sequential computing includes the following elements:

1. primitive units of computation

2. composition rules for composing the primitive units of computation into executable and schedulable units

3. definition of address spaces which control the data to which the computation is applied.

Models of parallel computation add to the elements of sequential computation requirements for:

1. modes and topology of communication between units of computation which are executing in parallel and

2. modes and types of synchronization mechanisms.

An application is typically made up of algorithms and steps and phases within algorithms, each of which may be characterized by some model of computation. Units of computation and the modes and topology of communication are often disjoint between phases of an algorithm. These algorithms and applications must be mapped to a hardware architecture which also realizes one or more models of parallel computation including the specification of one or more modes of synchronization and/or topologies and modes of communication.

Design Concepts

This section outlines the design concepts which are used in the TRAC system to meet the requirements posed by the problem statement defined in the section entitled "Requirements for Effective Parallel Computing." The set of concepts which we will describe include configurability, partitioning, integrated I/O architecture, and multiple modes of communication.

Configurability is the ability to realize problem specific parallel architectures from a collection of resources. Partitioning is the separation of resource configurations which gives conflict free execution. An integrated I/O architecture constructs a solution for external storage as a part of the design of the memory architecture of the system. Multiple modes of communication refers to the use of both circuit switching and packet switching as modes of data movement through the components of the architecture.

Configurability

Configurability in TRAC is founded on the dynamic establishment of circuits in an interconnection network. The interconnection network used in the current incarnation of TRAC is a two-sided SW Banyan network [GOK73,LIP80]. Processors are placed at the apex nodes of the switch and memory-I/O units at the bases of the switch. Figure 1 is a schematic of a 4-processor 9-memory-I/O configuration of TRAC. The architecture of TRAC and the properties of the banyan network have been well described in the literature [SEJ80,PRE80,LIP80]. We will focus on the properties which result from this architecture.

Three types of circuits are implemented. All have the shape of trees. The trees rooted at the apex nodes link one processor to many memory-I/O nodes and realize an address space for the processor. There are two types of trees with roots at the base of the network. An instruction tree couples a set of processors to give an SIMD architecture. A switchable memory tree establishes a family of circuits potentially linking many processors to a single memory-I/O unit. Only one circuit in this family may be active at any given time. Figure 2 illustrates the formation of such circuits for the 4-processor 9-memory-I/O configuration. It is straightforward to see that this capability for configuring architectures from resources through the establishment of circuits in the interconnection network supports realization of almost arbitrary parallel architectures.

Partitioning

Partitioning is again founded on the establishment of circuits in the network. Partitioning follows from the fact that a given resource element interacts on a cycle by cycle basis only with the resource elements to which it is joined by active circuits. The construction of a configuration from a collection of resources can thus be said to partition these resources.

Integrated I/O Architectures

The memory unit at each base node of the network may also have attached a self-managing secondary memory unit (SMSM) [RAT81,RAT83]. The SMSM is an object-oriented storage device. It implements associative searching by name and thus implements a local directory for the objects it has stored. The storage element for a SMSM may range from additional RAM to a large disk. An SMSM extends the address space of a memory unit to include the objects stored in the SMSM.

Multiple Modes of Communication

Efficient parallel formulations of problems and/or algorithms must minimize communication between streams of computation which are proceeding in parallel. Parallel architectures are, however, only of interest when there must be some cooperation and thus communication between the parallel streams of computation. The ideal communication system will have zero latency, infinite bandwidth and realize arbitrary topologies of communication. These are the properties of a paracomputer [SCH80]. We argue that realization of these properties in an interconnection network based architecture can be best approached through the use of both circuit and packet modes of communication. The TRAC interconnection network does realize both modes of data movement. Circuit based movement of data is based upon selective activation of circuits in the switchable trees. If one processor deactivates a circuit in a switchable tree and another processor activates a circuit in that tree then the entire contents of that memory are moved between the address spaces of the two processors.

TRAC also implements a complete packet switching capability. A processor may direct any memory to which it has an active circuit to send a data packet selected from its contents to any other processor. In fact, all processors may simultaneously transmit packets.

Effectiveness of TRAC as a Parallel Architecture

This section demonstrates how an architecture built on the design principles described preceding

realizes the requirements defined in the first section of the paper.

Realization of Multiple Models of Computation

It is obvious that TRAC can implement multiple models of parallel computation. There might, however, be some need to discuss the efficient implementation of actual computations. We will discuss implementation of communication in a separate sub-section. The TRAC architecture can incorporate almost any type or class of processor. The principal question to be asked is the efficiency of access to memory by an individual processor. Analyses of the switch node have shown that it is readily possible to construct nodes with delays of the order of a few nanoseconds. The length of the path between processors and memories is bounded by log n where n is the number of processors so that total delay will be not more than 10's of nanoseconds for even quite large numbers of processors. This is still much less than cycle time for memory cells appropriate for processors of considerable speed. It is also readily possible to include processor local caches in the TRAC design although we have not done so.

There are two other issues to be addressed. These are the costs of configuration and reconfiguration and the problems created by blocking in the network causing effective loss of resources. The TRAC architecture provides hardware support for the formation of configurations [RAT80,JEN81,JEN82]. Actual establishment of configurations can be accomplished in microseconds. The problems of allocation of resources to form configurations has been extensively studied by modeling ad simulation [DEG81a,PRE81]. These studies have consistently shown that rather simple algorithms are apparently adequate for reasonable levels of resource usage and rates of change of configurations.

Communication and Data Movement

The combination of circuit switching and packet switching mechanisms for communication implemented by TRAC synergistically combine to give very high performance over a spectrum of requirements for data movement. The switchable trees give a mechanism for very high bandwidth communication. The deactivation of one circuit and the activation of another in a switchable tree can be accomplished in a few memory cycle times. The result is the transfer between process address space of the entire contents of a memory-I/O cell which may include megabytes of direct access memory as well as executable memory.

Packets implement a mechanism of arbitrary topology but lower bandwidth. Packet based communication can be used where the establishment of circuits has an unacceptably high resource cost or is simply not possible.

Studies of image processing [YAS83] algorithms have suggested the power of combining the two modes of communication for interconnection network based architectures.

The I/O Problem

A many processor parallel architecture must face the problems of distributing data to processors, maintaining a directory structure, maintaining consistency and developing the necessary bandwidth of I/O. TRAC provides the capability of placing a self-managing secondary memory (SMSM) on every memory node. The overhead of maintaining a central directory is not required since the SMSM's implement local directory services which can be combined to give a distributed directory. Parallel directory searches can be executed by having each SMSM initiate a search on its own portion of the distributed directory. The availability of an I/O port at each memory site gives an enormous opportunity for parallel structuring of I/O processing. Thus TRAC attains I/O bandwidth through parallelism. The external data necessary for execution of a given process can be stored on the SMSM's local to that process. Thus complex data movements to complete I/O are avoided.

Support for Software

Resource management in an environment of thousands of processors, memories and I/O devices is an unsolved problem. The overhead of resource management in resource sharing systems tends to rise in a greater than linear fashion with the degree of competition for the resources and the mediation of conflict for resources. TRAC avoids this problem since sharing is all explicit. Each configuration of resources organized into a given computer architecture executes independently of all other configurations because the establishment of circuits partitions the resources. The processes interact only when interactions are programmed as a part of the computation. DeGroot [DEG81b] has discussed resource management for TRAC using the hardware support provided.

The structure of the operating system for TRAC has been described elsewhere [BRO82]. Figure 3 is a schematic of the operating system for TRAC.

The structure of this operating system is hierarchical with the functionality partitioned on a job-by-job basis. This structure in effect creates a local operating system for each job. The local operating systems interact only when job configurations are established and altered. This proposed implementation has a growth in resource management overhead which is linear in the number of jobs being executed on the entire architecture.

Extensibility

The concept base for TRAC can be effective for a spectrum of configurations of processors from a small to moderate number of relatively fast processors to a large number of slower processors. Estimates made of loss of effectiveness through

conflict and management overhead [JEN82] suggest that a configuration of a thousand processors each in the class of a few MIPS should be efficiently realizable in both hardware and software.

The memory and I/O system is modular. Large memories are realized by combining smaller units through circuit establishment. Distribution of I/O capacity with memory units prevents the development of a data movement problem for large numbers of processors.

The extensibility of TRAC is limited by the cost growth of n log n for the switch nodes and by the density of wires in this switch structure. A one-sided banyan [GOK79] has only linear growth in switch nodes as n becomes large and may allow for still larger configurations founded on the same principles.

Summary

The discussion given preceding argues that the design concepts of TRAC offer a cost effective basis for realizing very high performance parallel architectures. The practicality of these concepts has been demonstrated in the TRAC prototype now running in Austin and through numerous modeling and simulation studies.

Acknowledgements

This work was partially supported by grants from the National Science Foundation (Grant Number MCS-8116099), the Air Force Office of Scientific Research (Grant Number AFOSR 82-0091), and the Department of Energy (Grant Number DE-A305-81ER10987).

References

1. [GOK73] Goke, R. and Lipovski, G.J., "Banyan Networks for Partitioning on Multiprocessor Systems," Proc. 1st Symp. on Comp. Arch. 1, 1973, pp. 21-30.

2. [LIP80] Lipovski, G.J., Premkumar, U.V., Kapur, R.N., Malek, M. and Horne, P., "Design and Implementation of the Banyan Interconnection Network in TRAC," AFIPS Conf. Proc., May 1980, pp. 643-653.

3. [SEJ80] Sejnowski, M.J., Upchurch, E.T., Kapur, R.N., Charlu, D.P.S. and Lipovski, G.J., "An Overview of the Texas Reconfigurable Array Computer," AFIPS Conf. Proc., May 1980, pp. 631-641.

4. [PRE80] Premkumar, U.V., Kapur, R.N. and Lipovski, G.J., "Organization of the TRAC Processor-Memory Subsystem," AFIPS Conf. Proc., May 1980, pp. 623-629.

5. [RAT83] Rathi, B.D., "The Design and Performance Analysis of a Self Managing Secondary Memory," Ph.D. Dissertation, Department of Electrical Engineering, The University of Texas at Austin, December 1983.

6. [RAT81] Rathi, B.D., "Principles of Operation of TRAC's Self-Managing Secondary Memory," preliminary technical report TRAC 25, 1981.

7. [SCH80] Schwartz, J., "Ultracomputer," ACM TOPLAS 2, 1980, pp. 484-521.

8. [RAT80] Rathi, B.D., Tripathi, A. and Lipovski, G.J., "Hardwired Resource Allocators for Reconfigurable Architectures," Proc. Int. Conf. on Parallel Processing, August 1980, pp. 109-117.

9. [JEN81] Jenevein, R., DeGroot, D. and Lipovski, G.J., "A Hardware Support Mechanism for Scheduling Resources in a Parallel Machine Environment," Proc. 8th Int. Symp. on Comp. Arch., 1981, pp. 57-66.

10. [JEN82] Jenevein, R., DeGroot, D., Lipovski, G.J. and Browne, J.C., "A Control Processor for a Reconfigurable Array Computer," Proc. 9th Int. Symp. on Comp. Arch., Austin, Texas, 1982, pp. 81-89.

11. [BRO82] Browne, J.C. and Lipovski, G.J., "Reconfigurable Network Architectured Computer Systems: An Environment for Parallel Computing," Proc. Int. Workshop on High-Level Language Comp. Arch., Nov. 30-Dec. 3, 1982, pp. 40-49.

12. [YAS83] Yasrebi, M. and Browne, J.C., "A Comparison of Circuit Switching and Packet Switching for Data Transfer in Two Simple Image Processing Algorithms," Proc. 1983 ICPP, August 1983, pp. 25-28.

13. [GOK79] Goke, R. and Warren, W.L., "Linear/cost Banyan Interconnection Networks for Multi-Micro-Processor Systems," Proc. 3rd Rocky Mt. Symp. on Microcomputers, August 1979, pp. 61-88.

14. [DEG81] DeGroot, D., "Dynamic Mappings of Banyan Network Architectures," Ph.D. Dissertation, Department of Computer Sciences, The University of Texas at Austin, December 1981.

15. [PRE81] Premkumar, U.V., "A Theoretical Basis for the Analysis of Regular SW Banyans," Ph.D. Dissertation, Department of Electrical Engineering, The University of Texas at Austin, May 1981.

16. [DEG81b] DeGroot, D., Browne, J.C. and Malek, M., "A Resource Scheduling Problem for Reconfigurable Parallel Processing Systems," Proc. of Spring COMPCON, Feb. 1981, pp. 218-224.

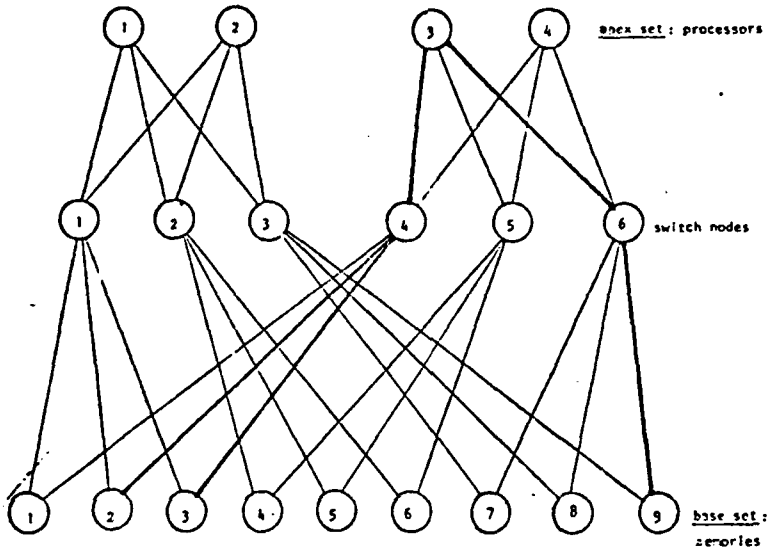


Figure 1

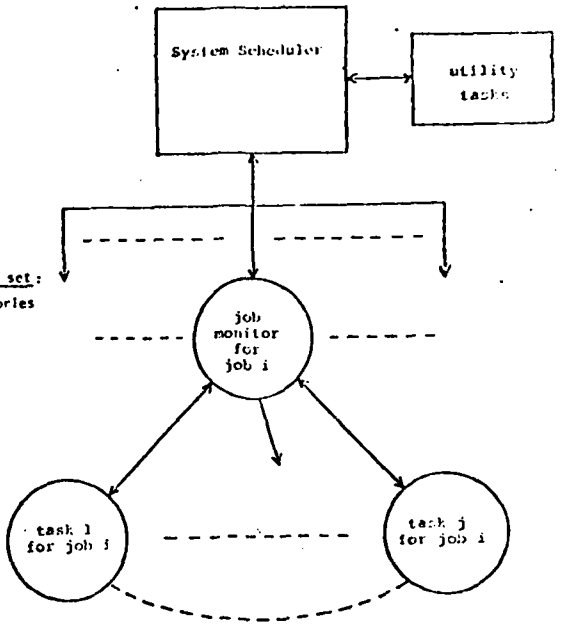


Figure 3: Schematic of Structure of TRACOS

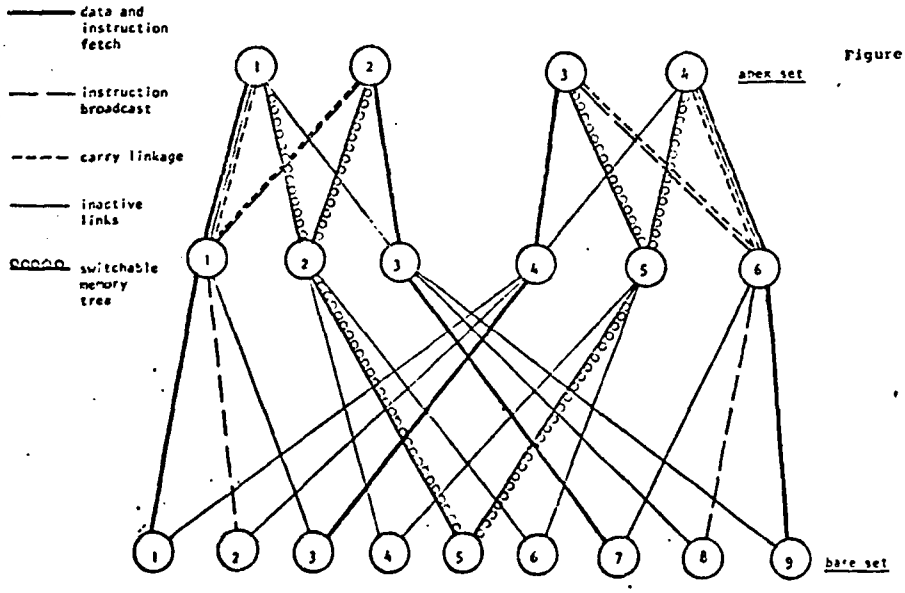


Figure 2: two LCS's with a switchable memory (node 5)

END

FILMED

3 -86

DTIC